HomebrewLab - Back-end

Tommaso Ballardini, Sirlan Fernandes, Federico Macchi

Giovedì, 06/07/2017

Indice

Capitolo 1

Introduzione

1.1 Note iniziali

Struttura report e note Il report è organizzato indicando, per ciascuna tecnologia, i motivi per i quali è stata adottata per lo sviluppo dell'applicazione *HomebrewLab*, dei benchmark e/o dei commenti aggiuntivi laddove necessari e, in alcuni casi, alcuni frammenti di codice a titolo di esempio. Sono state inserite anche delle immagini contenenti diagrammi o tabelle a corredo di quanto scritto. Infine, in fondo al documento, sono indicati tutti i riferimenti bibliografici inseriti. Il codice dell'intero progetto è disponibile sulla pagina GitHub del Prof. Florian Daniel.
Per la stesura del documento, è stato utilizzato LaTeX, mentre per la realizzazione delle immagini Adobe Photoshop CS6 e il software [`draw.io`] (applicazione web). Tutti i link cliccabili sono indicati tra parentesi `[ ]`.

1.2 L'idea

L'applicazione *La web app homebrewlab*, in breve *HomebrewLab*, è nata con lo scopo di aiutare gli utenti, seguendoli passo passo, nella creazione di birra artigianale nella propria abitazione. Esistono già svariate applicazioni simili, ma nessuna di queste riesce a unire in modo ottimale tutte le possibili funzioni di cui un *homebrewer*, sia alle prime armi, sia esperto può avere bisogno.

Le funzioni Grazie al supporto del Prof. Florian Daniel, esperto *homebrewer*, e ad una serie di riunioni abbiamo evidenziato quelle che ci sono sembrate le funzioni essenziali:

1. Gestione degli **ingredienti**
2. Gestione dell'**attrezzatura** e dei **processi** produttivi

3. Formulazione di ricette **pre-produzione** a partire da:
    - Ingredienti base
    - Una precedente ricetta, prodotta con lo stesso impianto
    - Una precedente ricetta, prodotta con un impianto differente
    - Una ricetta trovata online
4. Stampa di una **checklist** per la produzione
5. Assistenza per la:
    - Conversione di unità di misura
    - Preparazione dell'acqua
    - Preparazione starter
6. Gestione della ricetta **durante la produzione** con ricalcolo della stessa in funzione dell'effettivo andamento della produzione, quindi tenendo conto di eventuali variazioni di caratteristiche del prodotto
7. Creazione di una *dashboard*, o *lab*, privata per ogni utente
8. Possibilità di condividere contenuti tra utenti

1.3 Architettura

In figura 1.1 viene illustrata l'applicazione: essa è composta da due `moduli` principali, ognuno con una specifica funzione:

1. Client: composto da `dashboard` e `documentazione`, rappresenta tutto ciò vede l'utente. Ogni box indica una specifica pagina (o un insieme di pagine)
2. Server: è l'interfaccia tra ciò che vede l'utente e tutti i dati dell'applicazione. Ogni box rappresenta un gruppo di funzioni, dedicate a supportare un'operazione specifica.

I due `moduli` principali sono supportati da un modulo di autenticazione, indicato in figura come `layer di autorizzazione` che fornisce ad entrambi i moduli le funzioni base di autenticazione, identificazione e autorizzazione degli utenti che interagiscono con l'applicazione. La direzione delle frecce in figura, indica la direzione dei dati: una doppia freccia significa quindi un accesso in lettura e scrittura, mentre una freccia singola indica un accesso di sola lettura/scrittura.

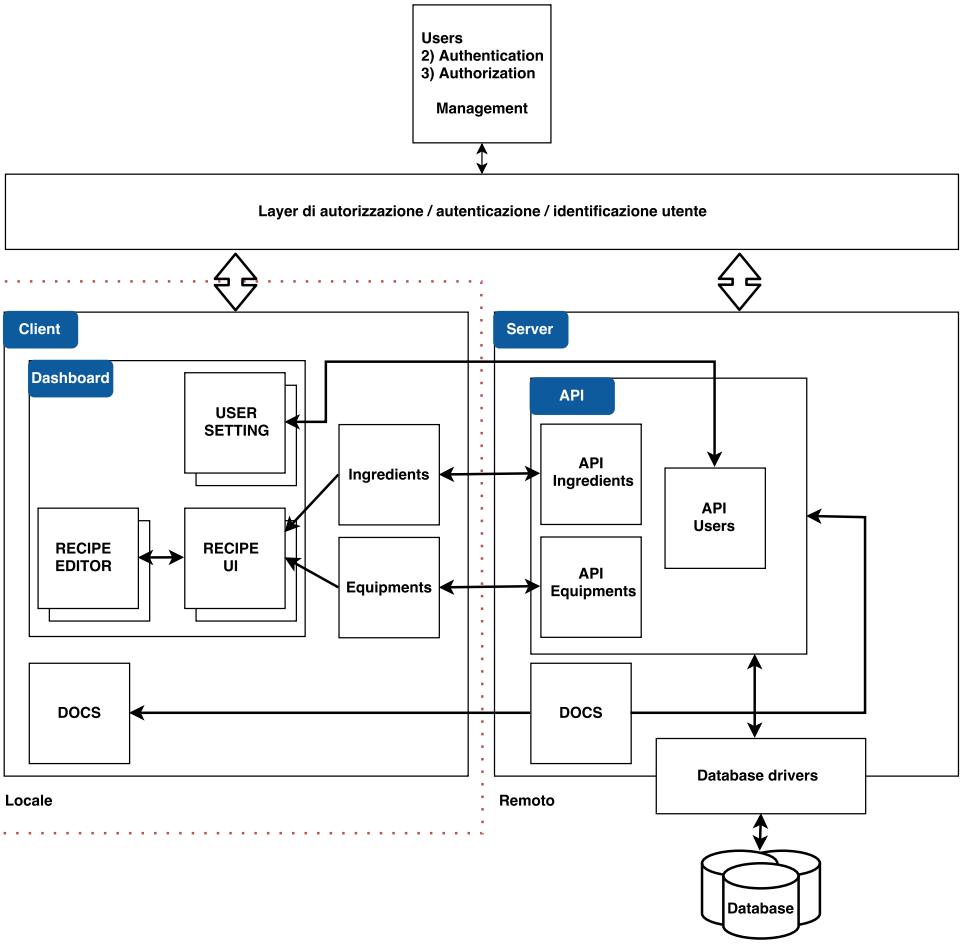

Figura 1.1: Overview dell'architettura generale dell'applicazione

Overview L'architettura scelta per l'applicazione è basata su un *fat-client* e un *fat-server*: il server gestisce tutta la parte di accesso autenticato ai dati dell'applicazione, mentre il client si occupa dell'elaborazione dei dati, tramite la libreria *Homebrewlib* appositamente sviluppata. Nel nostro caso, il client, pur essendo di tipo *fat* ha comunque bisogno di un accesso costante al server.

Implementazione Questa architettura è stata implementate utilizzando svariate tecnologie, tra cui:

- NodeJS: *core language* del server
- NoSQL: per la manipolazione dei dati
- HTML5 + CSS3 + VueJS: per la realizzazione del front-end

In figura 1.2 sono inserite le principali tecnologie, sia come linguaggi di programmazione che servizi esterni, utilizzate per l'implementazione dei moduli illustrati nel paragrafo introduttivo.

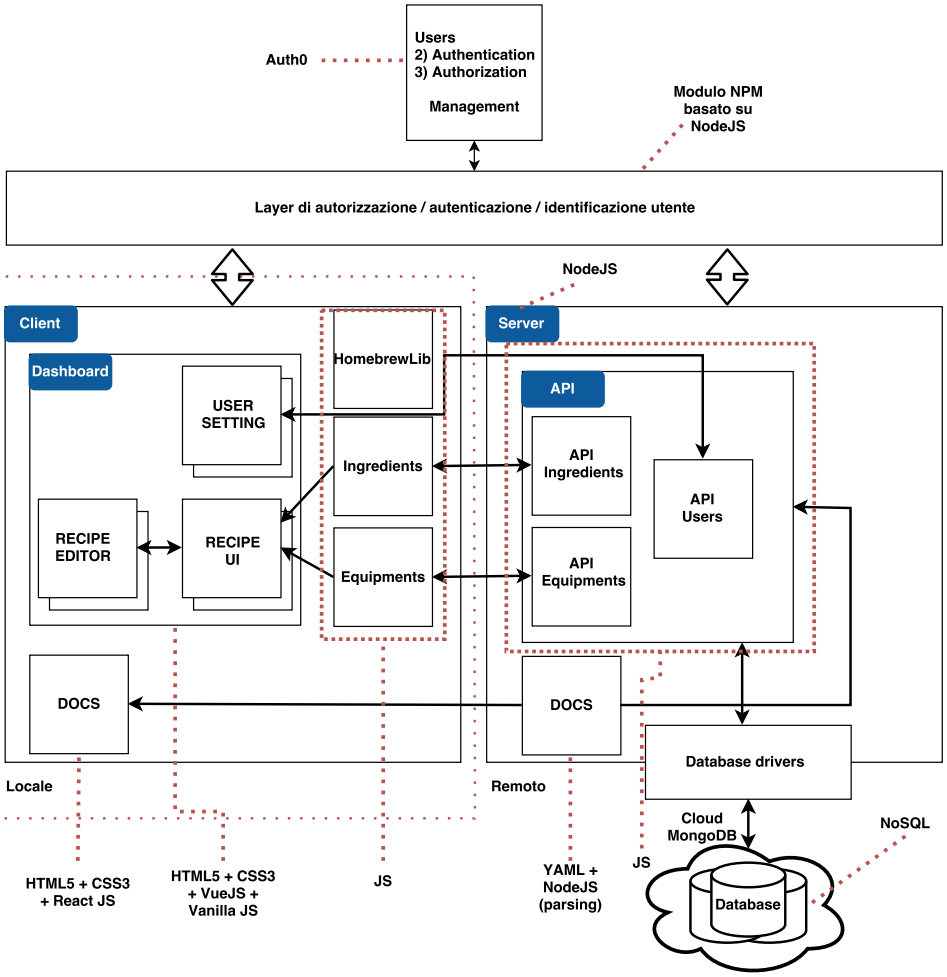

Figura 1.2: Implementazione reale dell'architettura in figura 1.1

Capitolo 2

Server

2.1 Hardware principale

GCC La soluzione adottata per il server, dopo svariati tentativi su differenti servizi, è stata la piattaforma cloud di Google, *Google Cloud Platform*, in breve *GCC* per diversi motivi:

- Costi: registrando gratuitamente un account sviluppatore si ottiene un credito di 300$ da spendere sulla piattaforma
- Flessibilità al 100%: la tipologia di macchina scelta, *Compute Engine* è un vero e proprio server virtuale con sistema operativo Linux. Da un lato ha richiesto parecchia manutenzione, nonché una buona conoscenza della riga di comando di Linux, però, d'altro canto, ha anche un costo decisamente inferiore rispetto alle macchine di Google *App Engine*, già impostate lo sviluppare software

Abbiamo stimato con il calcolatore online di Google[1] che, a parità di potenza di calcolo, una *APP Engine* costa circa il triplo al mese di una *Compute Engine*. Le altre due piattaforme, che sono state valutate, sono *AWS: Amazon Web Services* di Amazon, anch'essa con prova gratuita, e *Heroku*. La scelta finale è ricaduta sulla *GCC*, oltre che per le motivazioni sopra descritte, anche per la possibilità di accedere alle API di Google senza dover fare ulteriori configurazioni. Contrariamente a Heroku, Google offre anche la possibilità di installare dei propri certificati SSL e di utilizzare un dominio personalizzato senza costi extra.

2.1.1 Hardware di supporto: database

Come visto nell'introduzione, il database si trova invece su un server esterno alla *GCC*, precisamente sul Cloud di MongoDB, con lo scopo di ridurre il carico di lavoro della *Compute Engine* che così si deve occupare di gestire solamente le richieste dei client connessi e non deve eseguire un altro server come database.

2.2 Software

Definizione 1. *In computing, a solution stack or software stack is a set of software subsystems or components needed to create a complete platform such that no additional software is needed to support applications. Applications are said to "run on" or "run on top of" the resulting platform. Some definitions of a platform overlap with what is known as system software. For example, to develop an IT solution; in the case of a web application the architect defines the stack as the target operating system, web server, database, and programming language. Another version of a solution stack is operating system, middleware, database, and applications.[1] Regularly, the components of a solution stack are developed by different developers independently from one another. Some components/subsystems of an overall system are chosen together often enough that the particular set is referred to by a name representing the whole, rather than by naming the parts. Typically, the name is an acronym representing the individual components. [2]*

Stack Lo *stack* utilizzato, variante del *MEAN Stack* (MongoDB, Express JS, Angular JS, NodeJS), è composto da:

1. MongoDB
2. Express JS
3. Vue JS
4. Node JS

2.2.1 MongoDB

MongoDB è stato scelto principalmente per la flessibilità che assicura: *HomebrewLab* si è evoluta parecchio nel tempo, e MongoDB ha permesso di ridefinire lo schema dei dati diverse volte, senza dover apportare modifiche al codice già implementato.

Definizione 2. *MongoDB (da "humongous", enorme) è un DBMS non relazionale, orientato ai documenti. Classificato come un database di tipo NoSQL, MongoDB si allontana dalla struttura tradizionale basata su tabelle dei database relazionali in favore di documenti in stile JSON: JavaScript Object Notation con schema dinamico, rendendo l'integrazione di dati di alcuni tipi di applicazioni più facile e veloce. Rilasciato sotto una combinazione della GNU Affero General Public License e dell'Apache License, MongoDB è un software libero e open source. [3]*

Differenze con MySQL Contrariamente a MySQL, dov'è necessario fissare a priori l'architettura del database, tramite le *Primary Key* e *Foreign Key*, che forzano il modello dei dati dell'applicazione a rimanere costante nel tempo, MongoDB non prevede uno schema prefissato dei dati: non è necessario definire una chiave primaria unica, infatti il DBMS stesso si occuperà di generare un campo id, chiamato `_id`, univoco per ogni oggetto inserito, simile a questo: `507f1f77bcf86cd799439011`. Un'altra differenza sostanziale tra i due sistemi è la diversa procedura per l'interrogazione dei dati. Essendo MongoDB un database *NoSQL*, non utilizza la struttura classica di *MySQL*,

```
SELECT campi,
FROM database,
WHERE condizioni
```

ma una struttura molto più simile a un moderno linguaggio a oggetti, del tipo:

```
db.collezione.operazione();
```

Caratteristiche di base Come già detto, MongoDB è un database orientato ai documenti, ognuno dei quali è memorizzato in un formato simile al *JSON* chiamato *BSON: Binari JSON*. Il documento è fondamentalmente un *array*, che può a sua volta contenere altri *array* annidati. Un primo esempio è il seguente:

```
{
   "nome": "Dante",
   "cognome": "Alighieri",
   "lingue": ["italiano", "latino"],
   "opere": [
      {
         "titolo": "Divina Commedia",
         "iniziata": 1300,
         "versi": "endecasillabi"
      }
   ]
}
```

Le caratteristiche chiave del Cloud di MongoDB e generalmente dell'approccio *NoSQL* sono:

- Consente servizi di alta disponibilità , dal momento che la replicazione di un database (Replica Set) può avvenire in modo molto semplice e automatizzato.

- Garantisce la scalabilità automatica, ossia la possibilità di distribuire (Sharding) le collezioni in cluster di nodi, in modo da supportare grandi quantità di dati senza influire pesantemente sulle performance.

Operazioni base: inserimento Le collezioni vengono create implicitamente al primo utilizzo, quindi non è necessario farlo con un comando specifico. È sufficiente passare direttamente alla creazione di un documento:

```
db.utenti.insert({"name": "sirlan", "cognome": "fernandes",
    "email": "sirlan.fernandes@mail.polimi.it"});
```

Operazioni base: interrogazioni Per trovare tutti i documenti, inseriti in una specifica collezione, si utilizza l'apposita funzione `find();`

```
// Effettuo la query
db.utenti.find();
// Risultato:
{
   "_id" : ObjectId("5450e468efaef47248f4412c"),
   "nome" : "sirlan",
   "cognome" : "fernandes",
   "email" : "sirlan.fernandes@mail.polimi.it"
}
```

MongoDB ha autonomamente aggiunto un campo `_id`, che agisce come chiave primaria, poiché nell'operazione `insert` non era stato specificato. Un ObjectId è un numero pseudocasuale che ha un tasso di collisione infinitesimo (è cioè molto difficile generarne due uguali). Il metodo `find();` permette anche di eseguire interrogazioni complesse, basate sul valore di un singolo campo, sul valore di più campi in `AND` o in `OR` logico, e anche basandosi su `RegEx`: `RegularExpressions`.

Altre operazioni Le altre operazioni basilari sui dati, permettono il loro aggiornamento e la loro rimozione: l'aggiornamento dei dati è una combinazione della ricerca e dell'inserimento `db.utenti.update(campi da cercare, campi da aggiornare);`, mentre la remove è uguale alla ricerca, solo che al posto di restituire un documento, lo rimuove.

Validazione documenti La sintassi per la validazione di un documento, all'interno di una collezione, è molto semplice: consideriamo ad esempio di voler creare una *collection* "contacts" che ha come campi telefono e email, e vogliamo accettare solo email che hanno come dominio `mail.polimi.it`.

```
db.createCollection( "contacts",
```

```
    {
       validator: { $or:
          [
             { name:  { $type: "string", $required: "true"} },
             { phone: { $type: "string" } },
             { email: { $regex: /@mail.polimi\.it$/ } }
          ]
       },
       validationAction: "warn"
    }
 );
```

Il livello di validazione può essere impostato a diversi livelli, a seconda del risultato che si vuole ottenere: in questo caso "warn" indica che l'operazione di salvataggio o aggiornamento avrà sempre successo, anche se il documento non rispetta la validazione, ma verrà emesso un messaggio di log.

Validazione in HomebrewLab In *HomebrewLab* tuttavia, la validazione è interamente a carico di un modulo NPM di nome *Mongoose* che fornisce dei driver di alto livello per interagire con il database, basati sui driver ufficiali di MongoDB per NodeJS. *Mongoose* permette di definire attraverso Javascript dei modelli di dati e di creare contemporaneamente delle regole di validazione personalizzate. Consente anche di definire delle funzioni che vengono chiamate immediatamente prima o subito dopo una qualunque operazione sul database.

2.2.2 Express JS

Definizione 3. *Express è un framework per applicazioni web Node.js flessibile e leggero che fornisce una serie di funzioni avanzate per le applicazioni web e per dispositivi mobili. [4]*

Come per tutte le altre scelte, anche *express* js è stato scelto dopo una serie di analisi condotte sui vari framework: esso è stato ritenuto il miglior *trade-off* tra prestazioni e quantità di API disponibili rispetto a quelle standard di *NodeJS*. Il suo approccio è quello di costruire, attorno ai metodi già presenti in *NodeJS*, una sorta di *wrapper* con lo scopo di aggiungere nuove funzioni, senza nascondere o alterare quelle già presenti.

Definizione 4. *Express fornisce uno strato sottile di funzionalità di base per le applicazioni web, senza nascondere le funzioni Node.js che conosci e ami. [4]*

Sono stati analizzati diversi *benchmark* per avere una stima dell'effettiva velocità rispetto ad altri framework e rispetto a *NodeJS* puro.
In figura 2.1 sono evidenziate le *request/seconds* in due sistemi differenti:

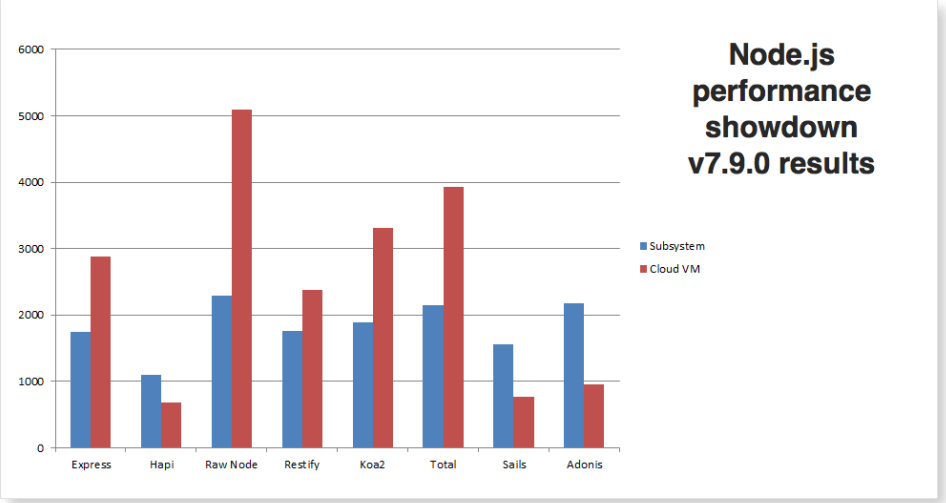

Figura 2.1: Benchmark sulle prestazioni di diversi framework. [5]

una generica *Virtual Machine* nel Cloud (Ubuntu 16.04 - 2GB Memory, 2 Cores) come la *GCC* e una generica *Virtual Machine* in un computer locale (Windows 10 PC - 32 GB RAM, i7-4790 CPU). Si evince immediatamente che *express js* ha ottime prestazioni, superato solamente da *Total js* e da *Koa2*. In particolare l'ultimo, Koa2, è stato sviluppato dallo stesso team di *express js*.

2.2.3 VueJS

Definizione 5. *Vue (pronounced /vju/, like view) is a progressive framework for building user interfaces. Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is very easy to pick up and integrate with other libraries or existing projects. On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with modern tooling and supporting libraries. [6]*

Come per le altre scelte, anche qui sono stati valutati diversi possibili *front-end frameworks* tra i più conosciuti:

- Angular JS
- React JS
- Vue JS
- JS standard (senza framework)

In figura 2.2 sono riportati i benchmark delle versioni più recenti dei framework sopra citati, nelle versione *keyed* e *non keyed*. Sotto alle due tabelle, sono inserite altre due piccole tabelle che indicano la quantità di memoria allocata in MBs.

Definizione 6. *Some frameworks like react, vue.js or angular allow to create a 1:1-relationship between a data item and a DOM node by assigning a "key" attribute. The other mode is "non-keyed" and this is what e.g. vue.js uses by default for lists. In this mode a change to the data items can modify DOM nodes that were associated with other data before. This can be more performant, since costly DOM operations can be avoided. [7]*

Dalla tabella in figura 2.2 si evince facilmente che Vue JS ha prestazioni molto simili al Javascript vanilla, in particolar modo nella creazione e rimozione di elementi dal *DOM, cioè l'insieme degli elementi che costituiscono una pagina web*, e "pesa meno" sulla memoria. (La tabella è disponibile anche come file allegato al presente report per una più semplice consultazione).

2.2.4 NodeJS

Definizione 7. *Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world. [9]*

Name	angular-v4.1.2-keyed	react-v15.5.4-keyed	vanillajs-keyed	vue-v2.3.3-keyed	Name	angular-v4.1.2-non-keyed	react-v15.5.4-non-keyed	vanillajs-non-keyed	vue-v2.3.3-non-keyed
create rows Duration for creating 1000 rows after the page loaded.	193.1 ±7.9 (1.4)	188.9 ±10.9 (1.4)	138.5 ±5.8 (1.0)	166.7 ±8.6 (1.2)	create rows Duration for creating 1000 rows after the page loaded.	198.8 ±10.1 (1.4)	187.9 ±9.0 (1.4)	137.8 ±6.2 (1.0)	168.1 ±10.1 (1.2)
replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations).	197.4 ±5.3 (1.3)	201.0 ±6.4 (1.4)	148.0 ±4.5 (1.0)	168.5 ±5.0 (1.1)	replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations).	59.6 ±4.8 (1.0)	77.5 ±2.3 (1.3)	64.0 ±7.9 (1.1)	68.0 ±3.4 (1.1)
partial update Time to update the text of every 10th row (with 5 warmup iterations).	13.0 ±4.5 (1.0)	16.5 ±2.3 (1.0)	14.1 ±4.7 (1.0)	17.3 ±2.9 (1.1)	partial update Time to update the text of every 10th row (with 5 warmup iterations).	13.2 ±3.8 (1.0)	16.3 ±2.6 (1.0)	12.4 ±3.2 (1.0)	17.1 ±1.5 (1.1)
select row Duration to highlight a row in response to a click on the row. (with 5 warmup iterations).	3.4 ±2.3 (1.0)	8.8 ±3.4 (1.0)	10.1 ±4.7 (1.0)	9.3 ±1.7 (1.0)	select row Duration to highlight a row in response to a click on the row. (with 5 warmup iterations).	4.1 ±2.0 (1.0)	8.2 ±3.2 (1.0)	8.4 ±4.3 (1.0)	10.5 ±2.3 (1.0)
swap rows Time to swap 2 rows on a 1K table. (with 5 warmup iterations).	13.4 ±1.0 (1.0)	14.7 ±0.9 (1.0)	11.4 ±1.1 (1.0)	18.3 ±1.5 (1.1)	swap rows Time to swap 2 rows on a 1K table. (with 5 warmup iterations).	8.7 ±0.8 (1.0)	11.6 ±1.2 (1.0)	8.0 ±0.8 (1.0)	14.5 ±1.1 (1.0)
remove row Duration to remove a row. (with 5 warmup iterations).	46.1 ±3.2 (1.1)	47.2 ±3.2 (1.1)	42.8 ±1.9 (1.0)	52.6 ±2.7 (1.2)	remove row Duration to remove a row. (with 5 warmup iterations).	34.4 ±2.6 (1.0)	53.1 ±2.4 (1.5)	35.8 ±3.6 (1.0)	42.7 ±2.0 (1.2)
create many rows Duration to create 10,000 rows	1946.0 ± 41.8 (1.5)	1852.4 ± 29.0 (1.4)	1331.1 ± 22.2 (1.0)	1587.5 ± 33.9 (1.2)	create many rows Duration to create 10,000 rows	1974.4 ± 37.4 (1.5)	1851.4 ± 34.1 (1.4)	1346.4 ± 30.1 (1.0)	1587.8 ± 31.6 (1.2)
append rows to large table Duration for adding 1000 rows on a table of 10,000 rows.	324.6 ±10.1 (1.1)	345.6 ±10.4 (1.2)	295.3 ±12.8 (1.0)	399.5 ±11.0 (1.4)	append rows to large table Duration for adding 1000 rows on a table of 10,000 rows.	328.2 ±11.3 (1.1)	351.1 ±13.5 (1.2)	294.1 ±10.1 (1.0)	399.8 ±10.1 (1.4)
clear rows Duration to clear the table filled with 10.000 rows.	379.9 ±11.3 (2.2)	398.4 ±8.2 (2.3)	174.8 ±4.2 (1.0)	254.5 ±5.0 (1.5)	clear rows Duration to clear the table filled with 10.000 rows.	393.8 ±9.9 (2.2)	393.0 ±9.5 (2.2)	175.3 ±4.9 (1.0)	253.9 ±5.6 (1.4)
startup time Time for loading, parsing and starting up	84.3 ±2.6 (2.1)	70.0 ±2.9 (1.7)	40.5 ±9.5 (1.0)	56.6 ±2.5 (1.4)	startup time Time for loading, parsing and starting up	83.3 ±2.3 (2.0)	68.3 ±1.9 (1.7)	41.0 ±8.7 (1.0)	58.2 ±2.5 (1.4)
slowdown geometric mean	**1.31**	**1.30**	**1.00**	**1.21**	slowdown geometric mean	**1.27**	**1.33**	**1.01**	**1.20**
Name	angular-v4.1.2-keyed	react-v15.5.4-keyed	vanillajs-keyed	vue-v2.3.3-keyed	Name	angular-v4.1.2-non-keyed	react-v15.5.4-non-keyed	vanillajs-non-keyed	vue-v2.3.3-non-keyed
ready memory Memory usage after page load.	4.8 ±0.0 (1.4)	4.5 ±0.1 (1.3)	3.4 ±0.0 (1.0)	3.8 ±0.0 (1.1)	ready memory Memory usage after page load.	4.8 ±0.0 (1.4)	4.5 ±0.1 (1.3)	3.4 ±0.0 (1.0)	3.8 ±0.0 (1.1)
run memory Memory usage after adding 1000 rows.	10.9 ±0.1 (2.7)	9.7 ±0.1 (2.4)	4.0 ±0.0 (1.0)	7.5 ±0.1 (1.9)	run memory Memory usage after adding 1000 rows.	10.9 ±0.1 (2.9)	9.7 ±0.1 (2.6)	3.8 ±0.0 (1.0)	7.5 ±0.1 (2.0)

Figura 2.2: Benchmark tra i framework front-end più popolari, VueJS, ReactJS, AngularJS e JS standard. A sinistra le versioni *non-keyed*, a destra quelle *keyed*. [8]

NodeJS ormai giunto alla versione 8.1.2 è considerabile un linguaggio abbastanza maturo per lo sviluppo di applicazioni professionali e pronte per un ambiente di produzione. Nel caso di *HomebrewLab*, *NodeJS* è sembrata la scelta migliore, in quanto essendo completamente asincrono e basato su eventi, è molto rapido nel rispondere alle richieste (HTTP), anche di molti client connessi contemporaneamente.

2.3 Autenticazione e identificazione con Auth0

Introduzione Per rendere l'autenticazione a HomebrewLab veloce e sicura abbiamo pensato di affidarci ad un servizio esterno, che garantisse alti standard di protezione dei dati, e fosse compatibile e facilmente adattabile alle nostre esigenze. Auth0 [`https://auth0.com`] corrisponde perfettamente a tali requisiti in quanto oltre ad essere gratuito fino ad un massimo di 7000 utenze attive (che hanno fatto almeno un accesso negli ultimi 30 giorni) è supportato da Partnership del calibro di Mozilla o Apple Safari. È inoltre documentato nel dettaglio per gran parte delle tecnologie in uso e consigliato per l'interazione con NodeJS ed Express e il mondo di Javascript in generale. La tecnologia consente l'accesso alle applicazioni tramite Social Login (Google e Facebook) o registrandosi tramite email e password.

2.3.1 Auth0, the new way to solve identity

Dashboard Una volta iscritti e autenticati in Auth0 è possibile accedere alla propria Dashboard dalla quale si ha il completo controllo tecnico e statistico delle utenze della nostra applicazione. È possibile visionare facilmente la densità dei login e la loro provenienza. Da qui si accede inoltre alle varie funzionalità della navbar laterale e alla documentazione. In figura 2.3 è riportata la pagina principale della *HomebrewLab*.

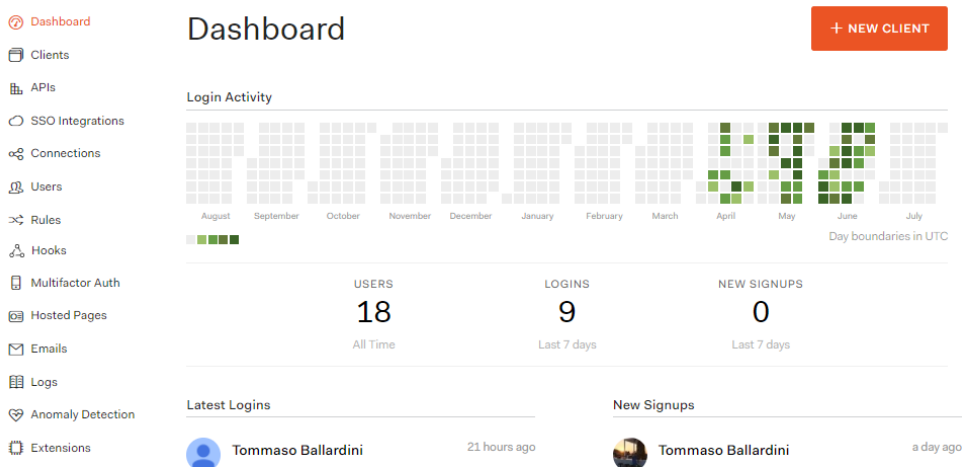

Figura 2.3: La *dashboard* di Auth0

Funzionalità La piattaforma offre diverse funzionalità per gestire efficientemente le utenze e le loro attività, alcune delle quali indispensabili per HomebrewLab:

- **Clients** Da ogni account di amministrazione è possibile impostare diversi client che gestiscano l'autenticazione a più applicazioni o a più funzionalità della stessa. Da qui è possibile impostare URL di `redirect` dopo il login/logout, e quella di `callback` da cui è possibile mandare richieste di vario tipo all'applicazione.

- **Users** Corrisponde ad una vera e propria UI di gestione degli utenti registrati all'applicazione; oltre che mostrare i dati relativi al login di ciascun utente e poterli modificare rende possibile bloccare qualche particolare utenza o eliminarla direttamente, il tutto tramite un'intuitiva interfaccia grafica.

- **Rules** Da questa sezione è possibile agire sull'autenticazione scrivendo degli script che vengono eseguiti nel momento in cui un utente accede all'applicazione, in modo da regolarne l'accesso. Nel nostro caso abbiamo implementato una *Rule* che riconoscesse gli utenti Admin tramite l'indirizzo email e che ne contrassegnasse il ruolo inviando un token specifico per garantire l'accesso a funzionalità differenti dell'applicazione.

Utilizzo Il servizio è assolutamente adattabile al progetto e le funzionlità sopra descritte possono essere utilizzate direttamente dall'applicazione integrandole nel *back-end* o comunicando con le API di *Auth0* che definiscono il grado di accesso dell'utente tramite il passaggio di un token e forniscono diverse informazioni sull'utenza.

2.3.2 Interazione e sicurezza

HTTPS *HomebrewLab* in quanto web app che raccoglie dati di utenti è stata pensata per aderire quanto più possibile ai recenti standard di sicurezza. Pertanto, tutto il traffico, sia entrante che uscente, è criptato tramite il protocollo SSL mediante una chiave a 2048bit. Un rapporto completo sulla sicurezza dei protocolli di comunicazione è disponibile nel file "report sicurezza.pdf" allegato al presente report [12]. Il firewall della *GCC* è configurato in modo da rispondere anche al traffico HTTP non criptato, che viene però immediatamente reindirizzato alla versione dell'applicazione criptata. Lo stesso vale per le richieste HTTP: se il protocollo è HTTPS esse vengono accettate, altrimenti vengono convertite in una richiesta di tipo GET su HTTPS.

I JWT Un JWT, acronimo di JSON Web Token è un particolare token composto da un `Header` che contiene tre campi: l'algoritmo usato per firmare il token, il tipo di token, e un KID che serve per identificarlo, un `Payload` contenente i dati, e una `Signature`.

Definizione 8. *JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA.*

JWTs are compact: because of their smaller size, JWTs can be sent through a URL, POST parameter, or inside an HTTP header. Additionally, the smaller size means transmission is fast.
And self-contained: The payload contains all the required information about the user, avoiding the need to query the database more than once.

Il token di HomebrewLab Il token utilizzato da *HomebrewLab* è firmato tramite l'algoritmo RSA con una coppia di chiavi pubbliche/private generate a run-time per ogni login direttamente sui server Auth0. Questi token non vengono infatti mai salvati sui server dell'applicazione, tantomeno le password degli utenti. Una copia del token viene salvata all'interno del `sessionStorage` del browser dell'utente al momento del login, e viene inviato automaticamente dal *front-end* al *back-end* con ogni richiesta. La durata di un token è di un'ora esatta: scaduto il token, esso non sarà più valido e bisognerà effettuare nuovamente il login.

Identificazione utente Utilizzando un modulo NPM sviluppato appositamente per *HomebrewLab*, *UserToken*, il server NodeJS identifica ogni utente che chiama le API utilizzando questo token, inserito nell'`Authorization Header` di ogni richiesta HTTP: inoltre, analizzando il token riesce a capire che tipo di permessi ha l'utente. Attualmente, sono infatti disponibili due tipi di permessi: `Admin` e `User`. Se il token non è presente nella richiesta, il server semplicemente la rifiuta. Se invece il token risulta essere stato alterato in qualche modo, per cui la validazione della firma fallisce, il server restituisce il codice `401`: `Unauthorized` e blocca qualunque operazione.

```
curl
-X GET "https://homebrewlab.it/users"
-H "accept: application/json"
-H "Authorization: Bearer TOKEN UTENTE"
```

Nel frammento di codice si può vedere come viene effettuata una richiesta tipo al server: si nota subito il parametro `-H` che imposta l'header

`Authorization` inserendo il token dell'utente, preceduto dal termine *Bearer*, mentre il parametro `-X` imposta l'indirizzo del server e il tipo di richiesta.

Workflow autenticazione Nella figura 2.4 è mostrato il workflow del processo di autenticazione e richiesta di una risorsa alle API di *Homebrew-Lab*. Si noti che i passaggi 1-2-3 vengono eseguiti solo una volta, al momento del login dell'utente. Nel caso in cui il token non sia valido, il workflow si interrompe al punto 5 (significa che l'utente ha presumibilmente saltato i punti 1-2-3 di autenticazione).

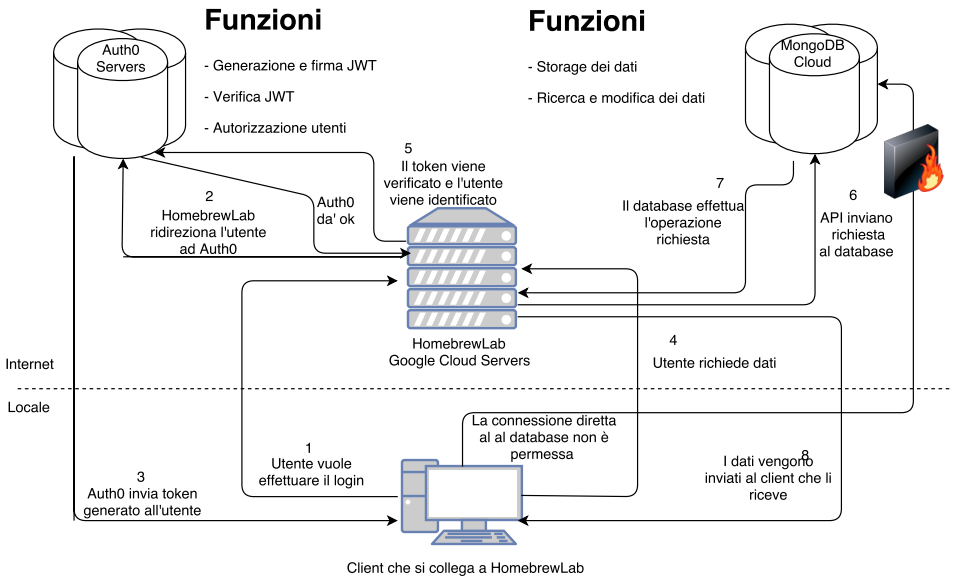

Figura 2.4: Workflow di autenticazione attraverso i servizi di *Auth0*

2.4 API

Definizione 9. *Con application programming interface (in acronimo API, in italiano interfaccia di programmazione di un'applicazione), in informatica, si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. Spesso con tale termine si intendono le librerie software disponibili in un certo linguaggio di programmazione. [14]*

API implementate Sono state implementate alcune API, tutte con le medesime funzioni, secondo il paradigma *REST*:

Definizione 10. *REST, REpresentational State Transfer, is an architectural style that abstracts the architectural elements within a distributed hypermedia system. A REST web service follows four basic design principles:*

1. *Use HTTP methods explicitly*
2. *Be stateless*
3. *Expose directory structure-like URIs*
4. *Transfer XML, JavaScript Object Notation (JSON), or both.*

Ulteriori dettagli sul paradigma REST sono disponibili sul qui [15].

Verbi HTTP I tipi di richiesta a cui il server risponde, o meglio, gli *HTTP Verbs* implementati sono:

- GET /endpoint: ritorna 1 o più documenti
- GET /endpoint/**parametro**: ritorna 1 documento corrispondente al parametro
- POST /endpoint: crea un nuovo documento
- PUT /endpoint/**parametro**: aggiorna il documento corrispondente al parametro
- DELETE /endopoint/**parametro**: elimina il documento corrispondente al parametro

Tutti gli endpoint indicati processano e rispondono solamente tramite documenti JSON. Le risposte, oltre a contenere il documento JSON dove applicabile, contengono anche un messaggio utile per fini di debug, un campo `HTTPStatusDescription` con una breve descrizione dello status della richiesta e un campo `HTTPStatus` che indica lo status della richiesta, secondo le specifiche HTTP. Maggiori informazioni e specifiche del protocollo HTTP sono disponibili qui [`https://tools.ietf.org/html/rfc2616`]. [16]

Documentazione La documentazione delle API è stata realizzata con *Swagger* qui: [`https://homebrewlab.it/docs`]. Mediante il pacchetto NPM *Swagger JSON-Doc* dei commenti annotati con `@swagger` e scritti nel linguaggio `YAML` vengono convertiti in un file JSON, visibile qui: [`https://homebrewlab.it/swagger.json`] che è poi trasformato in una pagina HTML dinamica grazie a *Swagger UI*. Da questa pagina, è possibile provare le funzioni delle API, previa autorizzazione tramite l'apposito pulsante: in questo modo è stato possibile scrivere la documentazione delle API e contemporaneamente testarle. La documentazione di swagger si trova sul sito ufficiale dedicato qui: [17].

2.5 Collegamento al Front-End

Autenticazione L'autenticazione dell'utente è realizzata esclusivamente tramite Javascript *front-end*, cosa generalmente non raccomandata, in quanto l'utente finale può potenzialmente modificare il codice Javascript eseguito sulla sua macchina al fine di apparire come autenticato. Per cui, allo scopo di rendere più resistente il meccanismo di login è stato inserito un ulteriore layer di sicurezza:

1. un apposito tag di *HTML5*, `<noscript>` verifica che l'utente abbia attivato Javascript
2. uno script js invia il token ai server di Auth0
3. se la risposta è `200`, quindi positiva, l'utente è considerato autenticato, altrimenti viene reindirizzato alla pagina di login iniziale.

Capitolo 3

Conclusioni

L'applicazione *HomebrewLab* è online e disponibile al seguente indirizzo [`https://homebrewlab.it`]. Effettuando il login tramite Google, Facebook o username-password, è possibile accedere al *Lab*, cioè la *dashboard* utente e visualizzare parte della UI già realizzata, come, ad esempio, la ricetta tabulare accessibile dal menu ricetta. In futuro, sarà disponibile una versione mobile più leggera e con meno funzioni, e probabilmente una versione desktop.

Bibliografia

[1] Google Cloud Platform
Platform Pricing Calculator
[https://cloud.google.com/products/calculator/]
(Ultima consultazione: 28-06-2017)

[2] Wikipedia, l'enciclopedia libera,
Solution Stack [https://en.wikipedia.org/wiki/Solution_stack]
(Ultima consultazione: 28-06-2017)

[3] Wikipedia, l'enciclopedia libera
MongoDB [https://it.wikipedia.org/wiki/MongoDB]
(Ultima consultazione: 28-06-2017)

[4] expressjs.com
Il framework express [http://expressjs.com/it/]
(Ultima consultazione: 28-06-2017)

[5] Raygun website
Node.js performance 2017: v7.9.0 vs. Hapi, Express.js, Restify and Koa and more by Alex Ogier [https://raygun.com/blog/node-js-performance-2017/]
(Ultima consultazione: 28-06-2017)

[6] Vue js website
Introduction to Vue JS - What is Vue.js? [https://vuejs.org/v2/guide/]
(Ultima consultazione: 28-06-2017)

[7] Stefan Krause blog
JS web frameworks benchmark keyed vs. non-keyed
[http://www.stefankrause.net/wp/?p=342]
(Ultima consultazione: 28-06-2017)

[8] Stefan Krause online blog
Results for js web frameworks benchmark - round 6
[http://www.stefankrause.net/js-frameworks-benchmark6/webdriver-ts-results/table.html]
(Ultima consultazione: 28-06-2017)

[9] NodeJS website
Homepage [https://nodejs.org/en/]
(Ultima consultazione: 28-06-2017)

[10] Mongoose homepage [http://mongoosejs.com/]
Mongoose docs [http://mongoosejs.com/docs/guide.html]
(Ultima consultazione: 28-06-2017)

[11] Nuove feature delle Promise [http://es6-features.org/#PromiseUsage]
L'oggetto Promise by Mozilla.org [https://developer.mozilla.org/it/docs/Web/JavaScript/Reference/Global_Objects/Promise]
(Ultima consultazione: 28-06-2017)

[12] SSL Report homebrewlab.it [https://www.ssllabs.com/ssltest/analyze.html?d=homebrewlab.it&hideResults=on] (il test potrebbe impiegare qualche minuto ad eseguirsi)
SSL labs homepage [https://www.ssllabs.com/index.html]
(Ultima consultazione: 28-06-2017)

[13] Homepage di auth0 [https://auth0.com/]
Features [https://auth0.com/docs/overview]
(Ultima consultazione: 28-06-2017)

[14] Wikipedia, l'enciclopedia libera
API [https://it.wikipedia.org/wiki/Application_programming_interface]
(Ultima consultazione: 28-06-2017)

[15] IBM developerWorks
learn SOA and webservices[https://www.ibm.com/developerworks/webservices/library/ws-restful/]
(Ultima consultazione: 28-06-2017)

[16] Hypertext Transfer Protocol – HTTP/1.1 [https://tools.ietf.org/html/rfc2616]
(Ultima consultazione: 28-06-2017)

[17] Swagger homepage [http://swagger.io/]
(Ultima consultazione: 28-06-2017)