Manuale Sviluppatore

Gruppo SteakHolders — ***Progetto MaaP***

Informazioni sul documento

Versione	3.0.0
Redazione	Enrico Rotundo, Gianluca Donato, Luca De Franceschi
Verifica	Federico Poli
Approvazione	Serena Girardi
Uso	Esterno
Distribuzione	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Gruppo SteakHolders
	CoffeeStrap

Descrizione

Il presente documento é il manuale per lo sviluppatore del framework MaaP.

Registro delle modifiche

Versione	**Data**	**Persone coinvolte**	**Descrizione**
3.0.0	2014-03-22	Serena Girardi (Responsabile)	Approvazione
2.2.0	2014-03-22	Federico Poli (Verificatore)	Verifica dell'incremento
2.1.1	2014-03-22	Luca De Franceschi (Amministratore)	Incremento
2.1.0	2014-03-22	Federico Poli (Verificatore)	Verifica della correzione
2.0.1	2014-03-20	Enrico Rotundo (Amministratore)	Correzione post-revisione
2.0.0	2014-03-13	Luca De Franceschi (Responsabile)	Approvazione.
1.1.0	2014-03-12	Federico Poli (Verificatore)	Verifica.
1.0.5	2014-03-09	Enrico Rotundo (Amministratore)	Stesura Glossario.
1.0.4	2014-03-07	Gianluca Donato (Amministratore)	Stesura Configurazione nuovo progetto.
1.0.3	2014-03-07	Gianluca Donato (Amministratore)	Stesura Requisiti di sistema.
1.0.2	2014-03-06	Enrico Rotundo (Amministratore)	Stesura MaaP Framework.
1.0.1	2014-03-03	Gianluca Donato (Amministratore)	Creato documento e stesa introduzione.

Indice

1 Introduzione

1.1 Scopo del documento

Questo documento intende descrivere i processi e le procedure da applicare per sfruttare al meglio le potenzialità del framework MaaP.

1.2 Scopo del prodotto

Lo scopo del progetto è la realizzazione di un *framework*$_G$ per generare interfacce web di amministrazione dei dati di *business*$_G$ basato su *stack*$_G$ *Node.js*$_G$ e *MongoDB*$_G$. L'obiettivo è quello di semplificare il processo di implementazione di tali interfacce che lo sviluppatore, appoggiandosi alla produttività del framework MaaP, potrà generare in maniera semplice e veloce ottenendo quindi un considerevole risparmio di tempo e di sforzo. Il fruitore finale delle pagine generate sarà infine l'*esperto di business*$_G$ che potrà visualizzare, gestire e modificare le varie entità e dati residenti in *MongoDB*$_G$. Il prodotto atteso si chiama *MaaP*$_G$ ossia *MongoDB as an admin Platform*.

1.3 Glossario

Ogni occorrenza di termini tecnici, di dominio e gli acronimi sono marcati con una "G" in pedice.

2 Requisiti di sistema

Il framework MaaP è compatibile con tutti i sistemi operativi supportati da `Node.js` e dal *Node Packet Manager*$_G$. I browser supportati sono la versione 30.0x o superiore di Chrome e la versione 24.x o superiore di Firefox.

È necessaria l'installazione di `Node.js` versione $\geq 0.10.0$[1] e del Node Packet Manager NPM versione $\geq 1.3.0$[2] per l'utilizzo del framework. Ulteriori dipendenze da librerie verranno risolte automaticamente al momento dell'installazione.

Per il completo funzionamento dell'applicazione web è richiesta la connessione ad un database MongoDB per l'autorizzazione e ad un database MongoDB per la gestione dei dati *business*$_G$. L'installazione e configurazione di tali database non viene effettuata dal framework MaaP.

Per l'installazione è richiesto l'utilizzo di un terminale da cui sia possibile eseguire il Node Packet Manager ed è richiesta una connessione ad Internet.

[1] Scaricabile da http://nodejs.org/dist/v0.10.0/
[2] Scaricabile da https://github.com/npm/npm/archive/v1.3.0.zip

3 Primo utilizzo

Vengono riportati in questa sezione i passi necessari per la creazione di una nuova applicazione MaaP, utilizzando il modello *"scaffold"*$_G$ fornito dal framework. I comandi descritti devono essere eseguiti da un terminale da cui sia possibile eseguire il Node Packet Manager ed è richiesta una connessione ad Internet.

3.1 Installazione

1. Installare il framework MaaP tramite il Node Packet Manager:
   ```
   $> npm install -g maap
   ```
2. Creare un'applicazione utilizzando il modello fornito dal framework MaaP: posizionarsi all'interno della cartella nella quale si vorrà creare l'applicazione ed eseguire il comando:
   ```
   $> maap create project <ProjectName>
   ```
   dove `<ProjectName>` corrisponde al nome del progetto da creare;
3. Installare le dipendenze dell'applicazione: posizionarsi all'interno della cartella ed eseguire il comando di installazione
   ```
   $> cd <ProjectName>
   $> npm install
   ```

3.2 Configurazione

1. Configurare l'applicazione: aprire con un editor di testo il file `config.js` generato assieme all'applicazione e configurarlo riferendosi alla sezione 4 di questo manuale.
2. Configurare le collections dell'applicazione, riferendosi alla sezione 5 di questo manuale. All'interno della cartella `collections` sono presenti dei file *DSL*$_G$ di esempio;

3.3 Avvio e accesso

1. Eseguire il server dell'applicazione: posizionarsi all'interno della cartella dell'applicazione ed eseguire il comando:
   ```
   $> npm start
   ```
2. Accedere all'applicazione da browser: aprire un browser (consigliati *Chrome* o *Firefox*) e accedere al server dalla pagina:
   ```
   localhost:3000/
   ```
   Notare che al primo utilizzo le collections sono vuote, in quanto è necessario popolare il database di prova.

3.4 Popolamento del database di prova

1. Posizionarsi all'interno della cartella `extra`:
   ```
   $> cd extra/
   ```

2. Eseguire lo script di popolamento del database utenti di prova:
   ```
   $> ./populate-users-db.sh -host localhost -port 27017 -db users
   ```
   Questo popolamento viene effettuato sul database impostato di default;
3. Eseguire lo script di popolamento del database delle Collection di prova:
   ```
   $> ./populate-data-db.sh -host localhost -port 27017 -db users
   ```
   Questo popolamento viene effettuato sul database impostato di default.

4 Configurazione nuovo progetto

4.1 Configurazione del back-end

Per configurare il back-end dell'applicazione generata dal framework bisogna modificare il file `config.js` presente nella cartella principale dell'applicazione. All'interno del file sono presenti due sezioni: "development" e "production", l'applicazione sceglie quale utilizzare in funzione del valore della variabile di ambiente `NODE_ENV`. Essi sono due configurazioni diverse: la prima riferisce a una configurazione di sviluppo, in cui è possibile impostare dei parametri in fase di sviluppo dell'applicazione; la seconda è invece la configurazione dell'applicazione che verrà resa pubblica. In ciascuna di esse è possibile impostare i seguenti parametri:

- **webServer**: un oggetto contenente
  - **port**: la porta su cui il server si metterà in ascolto (Integer);
  - **static**: il percorso assoluto della cartella contenente i file statici che il server dovrà servire al client (String);
- **userDB**: un oggetto contenente
  - **url**: l'indirizzo del database che l'applicazione deve utilizzare come database degli utenti (String);
- **usersPerPage**: di default settato a 5, indica il numero di utenti che si vogliono visualizzare per ogni pagina nella tabella degli utenti;
- **dataDB**: un oggetto contenente
  - **url**: l'indirizzo del database che l'applicazione deve usare come database dei dati (String);
- **smtp**: l' oggetto contenente la configurazione del servizio $smtp_G$ che verrà utilizzato dalla libreria `Nodemailer`. Per l'elenco completo di parametri configurabili riferirsi a https://github.com/andris9/Nodemailer;
- **resetPassword**: contiene la configurazione per il reset della password:
  - **tokenLife**: rappresenta il tempo di espirazione in millisecondi del reset password (Integer);
  - **link**: rappresenta il link da inviare per il reset della password (String);
- **collectionPath**: il percorso assoluto della cartella contenente i file di configurazione dsl che il server dovrà utilizzare come configurazione (String);
- **allowSignup**: contiene un valore booleano e indica se è possibile o meno utilizzare la funzionalità di registrazione (Boolean);
- **superAdmins**: contiene un array di utenti che di default dovranno essere settati come *superadmin*. Questo perché l'applicazione deve prevedere almeno un *superAdmin*. Questo array dovrà contenere oggetti con al loro interno l'email dell'utente e opzionalmente la password (String).

4.2 Configurazione del front-end

Per configurare il front-end dell'applicazione MaaP generata bisogna modificare il file:

`./app/scripts/config.js`

Al suo interno è possibile configurare le costanti di *Angular*$_G$. In particolare è possibile settare:

- **debug**: indica se mostrare i messaggi di errore in modo dettagliato oppure no (Boolean);
- **navBarCollections**: indica il numero di Collections da visualizzare nella barra di navigazione (Integer);
- **showSignup**: indica se mostrare o meno il pulsante di registrazione. È importante ricordarsi di disabilitare o riabilitare la funzionalità di registrazione anche nel back-end (Boolean);
- **reportLink**: indica il link al quale l'utente può segnalare bugs o problematiche relative all'applicazione. (String).

4.3 Struttura applicazione

L'applicazione generata dal framework presenta i seguenti file e cartelle:

- `ProjectName/`

  Contiene i file che verranno utilizzati come base di partenza (scaffold) per il progetto dello sviluppatore che utilizzerà il prodotto;

- `ProjectName/collections/`

  Contiene i file di configurazione delle collection di esempio utilizzate dall'applicazione scaffold. Di default è la cartella in cui devono essere salvati i file DSL;

- `ProjectName/app/`

  Contiene tutti gli script relativi al Frontend dell'applicazione;

- `ProjectName/app/scripts/`

  Contiene i file dell'applicazione AngularJS usata nel Frontend;

- `ProjectName/app/scripts/services/`

  Contiene i file dei service di AngularJS usata nel Frontend;

- `ProjectName/app/scripts/controllers/`

  Contiene i file dei controller di AngularJS usata nel Frontend;

- `ProjectName/app/view/`

  Contiene i file statici html usati dal Frontend dell'applicazione;

- `ProjectName/app/style/`

  Contiene i file statici CSS usati dal Frontend dell'applicazione;

- `ProjectName/app/bower_components/`

  Contiene i file delle librerie utilizzate dal Frontend dell'applicazione;

- `ProjectName/node_modules/`

  Contiene i file delle librerie utilizzate dal Backend dell'applicazione.

5 Configurazione di un file DSL

All'interno della cartella `collections/` sono presenti alcuni file `DSL` di esempio da cui trarre spunto. Come descritto in precedenza è possibile specificare manualmente su quale cartella andare a prelevare i propri file, intervenendo sulla configurazione dell'applicazione tramite il file `config.js`. Di default viene specificata questa cartella.

La configurazione di una *Collection*$_G$ avviene tramite la configurazione di un file DSL. Questa attività consiste nella semplice creazione o modifica di un file, quindi è possibile procedere alla configurazione tramite un qualsiasi editor di testo disponibile. La configurazione di base deve avere la seguente sintassi:

```
collection(
  name: "NomeCollection",
  label: "CollectionLabel",
  id: "CollectionId",
  weight: "CollectionWeight"
) {
  index(
    perpage: DocumentsPerPage,
    populate: AttributePopulate,
    sortby: "DefaultSort",
    order: "DefaultSortOrder",
    query: CollectionQuery
  ) {
    column(
      name: "AttributeName",
      label: "ColumnLabel",
      sortable: IndexSortable,
      selectable: IndexSelectable,
      transformation: TransformationFunction
    )
    ...
  }
  show(
    populate: AttributePopulate
  ) {
    row(
      name: "AttributeName",
      label: "RowLabel",
      transformation: TransformationFunction
    )
    ...
  }
}
```

L'applicazione, all'avvio del server, andrà a leggere il contenuto della directory impostata e preleverà sequenzialmente tutti i file con estensione `.dsl` contenuti al suo interno. Su questi file avverrà dunque un processo di *parsing*$_G$ da parte dell'interprete DSL, il quale si occuperà

di interfacciarsi con le API_G di MaaP e generare tutte le classi e i modelli necessari alla configurazione delle varie Collection indicate.

Se durante questo processo dovessero verificarsi degli errori relativi all'interpretazione dei file e all'esecuzione del codice prodotto essi verranno registrati e segnalati nell'applicazione.

All'interno del file è possibile definire funzioni e campi dati all'esterno del codice DSL, per poi riferirle all'interno di esso.

Ciascuna *espressione* prende in input una lista di parametri in stile javascript, ovvero: `nomeParametro: valoreParametro`. Alcuni parametri sono obbligatori, mentre altri sono facoltativi, e in caso vengano omessi vengono sostituiti con appropriati valori di default.

5.1 Configurazione di collection

Questa espressione indica come la Collection dev'essere configurata. Da essa deriveranno le configurazioni della *index-page* e *show-page*.

- **`name`** (required): questo parametro accetta una stringa e coincide con il nome della $Collection_G$ di riferimento sul database $MongoDB_G$;
- **`label`** (optional): questo parametro accetta una stringa e coincide con il nome che verrà visualizzato nella *index-page*. Se non specificato viene automaticamente inizializzato con il valore del parametro `name`;
- **`id`** (optional): questo parametro accetta una stringa e coincide con l'identificativo della URI_G della Collection. Di default questo parametro assume il valore del parametro `name`. Viene utilizzato se si vuole far puntare più configurazioni alla stessa collection su MongoDB;
- **`weight`** (optional): questo parametro accetta un valore intero e coincide con l'ordine di visualizzazione della Collection nella barra di navigazione dell'applicazione e nella lista delle collection della $Dashboard_G$. Se non specificato assume 0 come valore di default.

5.2 Configurazione di index

Questa espressione indica coma la index-page dovrà essere configurata. Da questa configurazione deriverà una precisa strutturazione e visualizzazione della pagina.

- **`perpage`** (optional): questo parametro accetta un valore intero maggiore di zero e coincide con il numero di Document che verranno visualizzate per ogni pagina. Se il numero totale di Document è maggiore del valore di questo parametro la index-page viene *paginata*. Se non specificato questo parametro prende come valore di default 50;
- **`populate`** (optional): questo parametro accetta una stringa e coincide con l'attributo esterno sul quale effettuare la funzione $populate_G$;
- **`sortby`** (optional): questo parametro accetta una stringa e coincide con l'attributo sul quale poter effettuare l'ordinamento di default nel caso in cui non sia specificata nessuna colonna con il parametro `sortable: true`;
- **`order`** (optional): questo parametro accetta una stringa con valore "`asc`" o "`desc`" che coincide con la tipologia di ordinamento del parametro `sortby`:

  - "`asc`" indica che l'ordinamento verrà fatto in modo *ascendente*;
  - "`desc`" indica che l'ordinamento verrà fatto in modo *discendente*;

- `query` (optional): questo parametro accetta un oggetto JSON con al suo interno i parametri e i valori sui quali effettuare la $query_G$ al database sulla Collection in questione.

5.3 Configurazione di column

Questa espressione indica la configurazione di una colonna della tabella della *index-page*.

- `name` (required): questo parametro accetta una stringa e coincide con il nome dell'attributo della collection di riferimento su MongoDB;
- `label` (optional): questo parametro accetta una stringa e coincide con il nome dell'intestazione della colonna nella tabella della *index-page*. Di default, se non specificato, assume il valore del parametro `name`;
- `sortable` (optional): questo parametro accetta un valore booleano che, se uguale a `true`, indica che la *index-page* può essere ordinata secondo questa colonna. Se non specificato assume `false` come valore di default;
- `selectable` (optional): questo parametro accetta un valore booleano che, se uguale a `true`, indica che l'elemento può essere selezionato tramite un link, il quale rimanderà alla relativa *show-page* del Document selezionato. Se non specificato assume `false` come valore di default;
- `transformation` (optional): questo parametro coincide con una funzione di trasforma: zione sull'elemento. La funzione deve restituire un valore, il quale sovrascriverà il valore dell'elemento estratto dalla query.

5.4 Configurazione di show

Questa espressione indica come la *show-page* dovrà essere configurata. Da questa configurazione deriverà una precisa strutturazione e visualizzazione della pagina.

- `populate` (optional): questo parametro accetta una stringa e coincide con l'attributo esterno sul quale effettuare la funzione $populate_G$.

5.5 Configurazione di row

Questa espressione indica la configurazione di una riga della tabella della *show-page*.

- `name` (required): questo parametro accetta una stringa e coincide con il nome dell'attributo della collection di riferimento su MongoDB;
- `label` (optional): questo parametro accetta una stringa e coincide con il nome dell'intestazione della riga nella tabella della *show-page*. Di default, se non specificato, assume lo stesso valore del parametro `name`;

- **`transformation`** (optional): questo parametro coincide con una funzione di trasforma: zione sull'elemento. La funzione deve restituire un valore, il quale sovrascriverà il valore dell'elemento estratto dalla query.

6 Esempi di configurazione di un DSL

Il codice seguente viene scritto solamente a titolo di esempio. Le porzioni di codice assumono che vi siano le opportune Collection e gli opportuni attributi dei Document su un database MongoDB esistente. L'esecuzione di questo codice potrebbe produrre degli errori di riferimento o comunque non avere un comportamento atteso in caso di mancanza di tali requisiti.

6.1 Esempio 1: lista di clienti

Si immagini di dover lavorare per un'azienda che vuole servirsi di MaaP per avere una vista costantemente aggiornata dei propri clienti.In particolar modo si vuole distinguere due *index-page* distinte:

- Una pagina in cui compaiono i clienti con un'età superiore ai 40 anni;
- Una pagina in cui compaiono i clienti con un'età inferiore ai 40 anni;

Di questi clienti si vuole sapere il nome, il cognome, l'indirizzo email, l'età e il numero di ordini effettuati, immaginando di avere un attributo della Collection di MongoDB chiamato *orders* contenente un array di ordini. Si vuole poter essere rimandati alla *show-page* del cliente tramite l'attributo *email*, e si vuole poter ordinare la tabella secondo il cognome, l'età e il numero di ordini. Nella *show-page* si vuole in aggiunta visualizzare anche l'indirizzo del cliente. Si vuole infine avere 20 elementi per pagina.

Il file DSL da produrre dovrebbe essere simile al seguente:

```
collection(
  name: "customers",
  label: "JuniorCustomers",
  id: "Junior",
  weight: "0"
) {
  index(
    perpage: 20,
    sortby: "surname",
    order: "asc",
    query: {age : { $lt : 40}}
  ) {
    column(
      name: "name",
      label: "Nome",
      sortable: false,
      selectable: false
    )
    column(
      name: "surname",
      label: "Cognome",
      sortable: true,
      selectable: false
```

```
    )
    column(
      name: "email",
      label: "Email",
      sortable: false,
      selectable: true
    )
    column(
      name: "age",
      label: "Eta",
      sortable: true,
      selectable: false
    )
    column(
      name: "orders",
      label: "# Ordini",
      sortable: true,
      selectable: false,
      transformation: function(val) {
        return val.length;
      }
    )
  }
  show() {
    row(
      name: "name",
      label: "Nome"
    )
    row(
      name: "surname",
      label: "Cognome"
    )
    row(
      name: "email",
      label: "Email"
    )
    row(
      name: "age",
      label: "Eta"
    )
    row(
      name: "address",
      label: "Indirizzo"
    )
    row(
      name: "orders",
      label: "# Ordini",
      transformation: function(val) {
        return val.length;
```

```
        }
      )
  }
}
```

Per quanto riguarda la configurazione della Collection degli utenti con età sopra ai 40 anni la sintassi è molto simile, ma è necessario apportare delle modifiche alle espressioni `collection` e `index` nel modo seguente:

```
collection(
  name: "customers",
  label: "Senior",
  id: "Senior",
  weight: "0"
) {
  index(
    perpage: 20,
    sortby: "surname",
    order: "asc",
    query: {age : { $gte : 40}}
  ) {
    ...
    ...
  }
  show() {
    ...
    ...
  }
}
```

6.2 Esempio 2: lista di prodotti

Si immagini di lavorare presso un'azienda fornitrice di prodotti che vuole avere a disposizione in tempi molto brevi uno strumento che fornisca una vista personalizzata dei prodotti fabbricati nell'anno corrente. L'azienda è interessata a visualizzare nella *index-page* il codice identificativo del prodotto, il nome del modello, le quantità disponibili in magazzino, il prezzo di vendita in euro, la data di produzione in formato americano. Inoltre vuole sapere se il prodotto è stato spedito oppure è ancora in magazzino. Nella *show-page* di un prodotto vuole visualizzare tutti i campi presenti. Infine vuole poter ordinare i prodotti per data di produzione e per quantità disponibili.

Il file DSL da produrre dovrebbe essere simile al seguente:

```
collection(
  name: "products",
  label: "Products - " + getCurrentYear(),
) {
```

```
  index(
    query: {
      productionDate.year: { $gt : getCurrentYear() }
    }
  ) {
    column(
      name: "_id",
      label: "Product id",
      selectable: true
    )
    column(
      name: "model",
      label: "Model"
    )
    column(
      name: "amount",
      label: "Quantity in stock",
      sortable: true
    )
    column(
      name: "price",
      label: "Price",
      transformation: euroFromDollar
    )
    column(
      name: "production_date",
      label: "Production date",
      transformation: getAmericanDate
    )
    column(
      name: "ship_date",
      label: "State",
      transformation: isShipped
    )
}

var euroFromDollar = function(val) {
  // converti val da dollari a euro
  return val;
}

var getCurrentYear = function() {
  // ritorna l'anno corrente
  return currentYear;
}

var getAmericanDate = function(date) {
  // converti date in data americana
  return americanDate;
```

```
55  }

    var isShipped = function(date) {
      var shipped = "";
      if (shipped === true) {
60      return "Shipped";
      }
      else {
        return: "Not shipped";
      }
65  }
```

7 Errori e malfunzionamenti

7.1 Lista degli errori

Durante l'esecuzione di un'applicazione MaaP possono venir segnalati degli errori. Di seguito viene riportata la lista di possibili errori che possono comparire. Notare che alcuni errori sono uguali tra loro nel contenuto ma diversi nel codice identificativo. Questo perché, nonostante rappresentino lo stesso errore, sono generati da file sorgenti diversi.

7.1.1 1xxx

- **1000**: Errore durante il login, username o password errati;
- **1001**: Accesso vietato, non si è autenticati;
- **1002**: Accesso vietato, non si dovrebbe essere autenticati;
- **1003**: Accesso vietato, livello utente sconosciuto;
- **1004**: Accesso vietato, è necessario avere il livello admin;
- **1005**: Accesso vietato, è necessario avere il livello superadmin;

7.1.2 2xxx

- **2000**: Utente non trovato, l'utente che si sta cercando tramite id non è stato trovato nel database;
- **2001**: Utente non trovato, l'utente che si sta cercando tramite email non è stato trovato nel database;
- **2002**: Utente non trovato, l'utente che si sta cercando tramite il token di reset password non è stato trovato nel database;
- **2003**: Errore di reset password, token non valido;
- **2004**: Pagina non trovata, il numero di pagina deve essere un intero positivo;

7.1.3 3xxx

- **3000**: Collection non trovata, la Collection che si sta cercando non è stata trovata;
- **3001**: Definizione di due Collection con la stessa id;

7.1.4 5xxx

- **5001**: Errore di interpretazione del DSL;
- **5002**: Errore di esecuzione del file DSL;

7.1.5 6xxx

- **6000**: Non trovato, la risorsa richiesta non esiste;

7.1.6 7xxx

- **7000**: Collection non trovata, la Collection richiesta non è stata trovata;

7.1.7 8xxx

- **8000**: Errore di esecuzione del DSL;
- **8001**: Collection id malformata;

7.1.8 9xxx

- **9000**: Accesso negato, la registrazione è disabilitata;
- **9001**: Accesso negato, non puoi eliminare un Super Admin o te stesso;
- **9002**: Accesso negato, non puoi creare un Super Admin;
- **9003**: Accesso negato, non puoi modificare un Admin o un Super Admin;
- **9004**: Credenziali non valide, devi specificare un'email e una password valida;

7.1.9 10xxx

- **10000**: Password errata, la password vecchia che hai inserito non è corretta;

7.1.10 12xxx

- **12000**: Document non trovato, il Document che hai richiesto non è stato trovato;

7.1.11 13xxx

- **13000**: Errore di esecuzione del DSL;

7.1.12 14xxx

- **14000**: Errore di esecuzione del DSL;

7.1.13 15xxx

- **15000**: Errore di esecuzione del DSL;

7.1.14 16xxx

- **16000**: Errore di esecuzione del DSL;

7.1.15 17xxx

- **17000**: Errore di esecuzione del DSL;

7.1.16 18xxx

- **18000**: Collection non trovata, la Collection che stavi cercando non è stata trovata.

7.2 Procedura di segnalazione di un errore

Se viene riscontrato un bug o un'anomalia all'interno del framework MaaP che non sia collegato alla lista appena presentata è necessario segnalarlo tramite l'apertura di una $issue_G$ su GitHub al seguente indirizzo:

https://github.com/steakholders/maap-dev/issues/new

Tale issue dev'essere il più possibile esplicativa e contenere il titolo e la descrizione della problematica. Gli sviluppatori del framework MaaP sono a completa disposizione per la risoluzione di issues relative al prodotto in questione e invitano tutti gli utilizzatori a contribuire alla segnalazione e all'individuazione di bug.

A Glossario

Angular: È un *framework*$_G$ open source Javascript, mantenuto da Google, utilizzato per creare componenti front-end.

API: Application programming interface, sono un insieme di procedure che un sistema software rende accessibile a terzi per interfacciarsi ad esso.

Business: Inteso come dominio di Business, l'insieme di tutti i dati che riguardano un dato campo.

Collection: In *MongoDB*$_G$ è un insieme di *documents*$_G$.

Dashboard: Pagina web che rappresenta lo stato corrente di un'applicazione con un interfaccia semplice ed immediata.

Document: Equivale ad una riga di una tabella. Vedi la definizione di Documents.

DSL: Domain Specific Language è un linguaggio di programmazione o un linguaggio di specifica dedicato a particolari problemi di un dominio o a una particolare tecnica di rappresentazione.

Issue: È un sinonimo di ticket contestualizzato nel sistema GitHub. Con il termine ticket si intende un'*unità*$_G$ di lavoro con cui apportare un miglioramento in un sistema. Un ticket può essere un *bug*$_G$, una richiesta di funzionalità, un compito, e così via.

Framework: Struttura di supporto su cui un applicativo può essere progettato. Un framework comprende librerie di codice, convenzioni di sviluppo e una serie di strumenti di supporto allo sviluppo.

MaaP: *Framework*$_G$ per generare interfacce web di amministrazione dei *dati di business*$_G$ basato su stack *Node.js*$_G$ e *MongoDB*$_G$.

MongoDB: Sistema gestionale di basi di dati non relazionale, orientato ai documenti, di tipo *NoSQL*$_G$. Il linguaggio utilizzato per la gestione dei dati è JavaScript, del quale sfrutta in particolare la notazione BSON.

Mongoose: Utilizzato per creare una struttura logica nei documenti di *MongoDB*$_G$.

Node.js: Piattaforma software utilizzata per creare applicazioni distribuite facilmente scalabili. Node.js utilizza JavaScript come linguaggio di scripting e gestisce le attese I/O in modo asincrono.

NoSQL: È un movimento che promuove sistemi software dove la persistenza dei dati è caratterizzata dal fatto di non utilizzare il modello relazionale, tipicamente usato dai database tradizionali. L'espressione NoSQL fa riferimento al linguaggio SQL, che è il più comune linguaggio di interrogazione dei dati nei database relazionali, qui preso a simbolo dell'intero paradigma relazionale. Questi archivi di dati tipicamente non richiedono uno schema fisso, evitano spesso le operazioni di unione e puntano a scalare orizzontalmente.

P

Parsing: Traducibile con analisi sintattica, è il processo atto ad analizzare uno stream continuo in input in modo da determinare la sua struttura grammaticale grazie ad una data grammatica formale.

Populate: È un processo di *Mongoose*$_G$ che permette di rimpiazzare automaticamente il path specificato nel *Document*$_G$ con *Document*$_G$ da altre *Collection*$_G$. A differenza dei normali database relazionali non ci sono join in MongoDB, e questa funzionalità permette un utilizzo analogo.

Scaffolding: In informatica è una procedura che automatizza la creazione di oggetti ed interfacce a partire da alcune semplici specifiche dettate dal programmatore.

Stack: Il termine stack o pila indica un tipo di dato astratto che viene usato in diversi contesti per riferirsi a strutture dati, le cui modalità d'accesso ai dati in essa contenuti seguono una modalità LIFO (Last In First Out), tale per cui i dati vengono estratti in ordine rigorosamente inverso rispetto a quello in cui sono stati inseriti.

URI: Acronimo di Uniform Resource Identifier, è una stringa che identifica univocamente una risorsa generica che può essere un indirizzo Web, un documento, un'immagine, un file, un servizio, ecc. e la rende disponibile tramite protocolli quali HTTP, FTP, ecc.