Manuale Sviluppatore

Gruppo SteakHolders — ***Progetto MaaP***

Informazioni sul documento

Versione	4.0.0
Redazione	Enrico Rotundo, Gianluca Donato
Verifica	Federico Poli
Approvazione	Luca De Franceschi
Uso	Esterno
Distribuzione	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Gruppo SteakHolders
	CoffeeStrap

Descrizione

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

Registro delle modifiche

Versione	**Data**	**Persone coinvolte**	**Descrizione**
4.0.0	2014-03-13	Luca De Franceschi (Responsabile)	Approvazione.
3.1.0	2014-03-12	Federico Poli (Verificatore)	Verifica.
3.0.5	2014-03-09	Enrico Rotundo (Amministratore)	Stesura Glossario.
3.0.4	2014-03-07	Gianluca Donato (Amministratore)	Stesura Configurazione nuovo progetto.
3.0.3	2014-03-07	Gianluca Donato (Amministratore)	Stesura Requisiti di sistema.
3.0.2	2014-03-06	Enrico Rotundo (Amministratore)	Stesura MaaP Framework.
3.0.1	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 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 MaaP Framework

2.1 A chi è rivolto questo manuale

Questo manuale è rivolto a tutti gli sviluppatori che necessitano di generare interfacce di amministrazione di dati in tempi ristretti. Per utilizzare questo prodotto non è strettamente necessario conoscere a fondo *Node.js* e *MongoDB* ma è altresì caldamente consigliato avere conoscenza e familiarità con il linguaggio *Javascript* e con l'amministrazione di sistema da linea di comando.

2.2 Perché utilizzare MaaP

3 Requisiti di sistema

Il framework MaaP è compatibile con tutti i sistemi operativi supportati dal framework `Node.js` e dal Node Packet Manager. 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 $\geq$ 0.8.0 e del Node Packet Manager NPM versione $\geq$ 1.3.0 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. 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.

4 Configurazione nuovo progetto

4.1 Generazione

Vengono riportate in questa sezione i passi necessari per la creazione di una nuova applicazione MaaP, utilizzando il modello "scaffold" 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.

1. Installare il framework MaaP tramite il Node Packet Manager:
   ```
   $> npm install -g maap
   ```
   oppure, se il framework viene fornito tramite repository git
   ```
   $> npm install -g maap
   ```
2. Creare un'applicazione utilizzando il modello fornito dal framework MaaP: posizionar: si all'interno della cartella nella quale si vorrà creare l'applicazione ed eseguire il comando
   ```
   $> maap create <ProjectName>
   ```
   Dove `<ProjectName>` corrisponde al nome del progetto che si vuole creare.
3. Installare le dipendenze dell'applicazione: posizionarsi all'interno della cartella dell'appli: cazione ed eseguire il comando
   ```
   $> npm install
   ```
4. Configurare l'applicazione: aprire con un editor di testo il file `config.js` generato assieme all'applicazione e configurarlo riferendosi alla sezione 4.2 di questo manuale.
5. Configurare le collection dell'applicazione, riferendosi alla sezione 5 di questo manuale.
6. Eseguire il server dell'applicazione: posizionarsi all'interno della cartella dell'applicazione ed eseguire il comando
   ```
   $> npm start
   ```

4.2 Configurazione

Per configurare l'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`. 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**: la path assoluta della cartella contenente i file statici che il server dovrà servire al client (String);
- **userDB**: un oggetto contenente
  - **uri**: l'indirizzo del database che l'applicazione deve usare come database degli utenti;

- **dataDB**: un oggetto contenente
  - **uri**: l'indirizzo del database che l'applicazione deve usare come database dei dati;
- **collectionPath**: la path assoluta della cartella contenente i file di configurazione dsl che il server dovrà utilizzare come configurazione (String);
- **smtp**: l' oggetto contenente la configurazione del servizio smtp che verrà utilizzato dalla libreria Nodemailer. Per l'elenco completo di parametri configurabili riferirsi a https://github.com/andris9/Nodemailer.

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 utilizzate dall'applicazione scaffold.
- **`ProjectName/app/`**
  Contiene i file relativi al Frontend dell'applicazione scaffold.
- **`ProjectName/app/view/`**
  Contiene i file statici html usati dal Frontend dell'applicazione scaffold.
- **`ProjectName/app/style/`**
  Contiene i file statici di stile usati dal Frontend dell'applicazione scaffold.
- **`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/bower_components/`**
  Contiene i file delle librerie utilizzate dal Frontend dell'applicazione scaffold.
- **`ProjectName/node_modules/`**
  Contiene i file delle librerie utilizzate dal Backend dell'applicazione scaffold.

5 Configurazione Collection

All'interno della cartella `../collection` andranno inseriti i file di configurazione delle Collection. L'applicazione infatti, all'avvio del server, andrà a leggere il contenuto della directory in questione e prelevare sequenzialmente tutti i file con estensione `.dsl` contenuti al suo interno. Questi file andranno poi ad essere interpretati dall'interprete DSL, il quale si occuperà di generare tutti i modelli necessari alla configurazione delle varie collection.

La configurazione di una collection avviene tramite l'editing di un file DSL. La configurazione base deve avere la seguente sintassi:

```
collection("collectionName","collectionReference", collectionQuery
    ) {
  index {
  }
  show {
  }
}
```

I parametri accettati dalla configurazione di una collection sono i seguenti:

`collectionName` corrisponde al nome della collection che verrà visualizzato nell'applicazione e deve essere di tipo String;

`collectionReference` corrisponde al nome effettivo della collection nel database MongoDB delle collections e deve essere di tipo String;

`collectionQuery` corrisponde ad un oggetto javascript tramite il quale può essere effettuata una query sulla collection.

Ciascuna index-page è composta da una serie di **`column`** e può essere configurata come segue:

```
index {
  column("columnLabel", "attributeReference", transformation,
selectable, sortable)
}
```

I parametri accettati dalla configurazione di una column di una index-page sono i seguenti:

`columnLabel` corrisponde al nome visualizzato nell'intestazione della colonna;

`attributeReference` corrisponde al nome effettivo dell'attributo dei document della collection nel database;

`transformation` corrisponde ad una funzione javascript che si occupa di effettuare una trasformazione sul valore dell'attributo. Tale funzione per effettuare l'effettiva trasformazione deve restituire un valore con il **`return`**;

`selectable` corrisponde ad un valore booleano che indica se l'elemento della colonna è selezionabile, ovvero se nell'index-page conterrà un link che rimanda alla corrispondente show-page del document;

`sortable` corrisponde ad un valore booleano che indica se la tabella è ordinabile secondo la colonna.

Ciascuna show-page è composta da una serie di `row` che può essere configurata come segue:

```
show {
  row("rowLabel", "attributeReference", transformation)
}
```

I parametri accettati dalla configurazione di una column di una index-page sono i seguenti:

`rowLabel` corrisponde al nome visualizzato nell'intestazione della riga;

`attributeReference` corrisponde al nome effettivo dell'attributo del document;

`transformation` corrisponde ad una funzione javascript che si occupa di effettuare una trasformazione sul valore dell'attributo. Tale funzione per effettuare l'effettiva trasformazione deve restituire un valore con il `return`;

Di seguito viene fornita un'immagine di esempio in cui viene configurata l'index-page e la show-page relativa ad una collection di *clienti*. Tale configurazione prende tutti i *clienti* di nazionalità italiana.

```
1  // Define the collection configuration
2  collection("myCustomer", "customer", {country: "Italy"} ) {
3      // Define the index-page configuration
4      index {
5          // Define the columns configurations
6          column("Id", "_id", null, true, )
7          column("Name", "customerName", null, false, true)
8          column("Surname", "customerSurname", null, false, true)
9          column("Email", "customerEmail", null, false, false)
10         column("Birth date", "customerBirthDate", function(date) { return date.toIsoDate(); }, false, false)
11     }
12     // Define the show-page configuration
13     show {
14         // Define the rows configurations
15         row("Name", "customerName", null)
16         row("Surname", "customerSurname", null)
17         row("Number of orders", "customerOrders", function(orders) { return orders.length; })
18         row("Mobile number", "customerMobileNumber", null)
19     }
20 }
```