



MongoDB as an admin Platform (MaaP)

Introduzione

Negli ultimi anni l'universo aziendale sta iniziando ad intravedere negli strumenti *open source* una risorsa estremamente preziosa, sia da un punto di vista economico, che da un punto di vista tecnologico. Questa convergenza è dettata in primo luogo dalla necessità di stare al passo con i ritmi e la flessibilità richiesti da esigenze di prototipazione del prodotto centrate sulla minimizzazione delle risorse da un lato, e la massimizzazione dell'efficienza dall'altro.

In particolare, la notevole mole di informazioni a disposizione delle piattaforme *online* dei nostri giorni ha impegnato progressivamente gli architetti *software* a rivedere e rivoluzionare i tradizionali paradigmi di archiviazione delle informazioni e lo stesso concetto di *storage* centralizzato e decentralizzato. Se fino a poco tempo fa un *database* relazionale, inserito all'interno di un'opportuna architettura, poteva gestire tale mole di dati, al giorno d'oggi si sente la necessità di utilizzare un nuovo paradigma di immagazzinamento dei dati: i database di tipo NoSQL (*Not only SQL*, <http://en.wikipedia.org/wiki/NoSQL>).

Un database NoSQL fornisce un sistema di immagazzinamento dei dati e di loro successivo recupero che richiede l'utilizzo di modelli meno vincolati e restrittivi rispetto ai database di tipo relazionale. Questo tipo di approccio permette una maggiore semplicità del processo di modellazione dei dati, una migliore propensione allo *scaling* orizzontale del database e un controllo maggiore della disponibilità dei dati.

Uno dei database di tipo NoSQL che si sta maggiormente affermando sul mercato come *standard de facto* (<http://www.mongodb.org/about/production-deployments/>) è MongoDB (<http://www.mongodb.org/>)

Interfacce di amministrazione dei dati di *business*

Qualsiasi applicazione che utilizzi un database per persistere i suoi dati, richiede operazioni di amministrazione e manutenzione di quest'ultimi. L'amministrazione sui dati può essere effettuata a due livelli:

1. **Amministrazione del database:** vengono effettuate operazioni direttamente sugli oggetti che rappresentano il database (tabelle, indici, viste) in modo tale da permettere un accesso veloce e consistente ai dati. Questa tipologia di amministrazione non si preoccupa di interpretare le informazioni in dati di *business*, ma si limita a interagire in modo agnostico con le entità del database

2. **Amministrazione dei dati di *business***: vengono effettuate operazioni sulle entità di *business* del proprio modello dati (ad esempio clienti, ordini, polizze, ...). In questo caso, le entità del database vengono interpretate nel modello di *business* richiesto e le operazioni di amministrazione possono richiedere anche l'intervento su più di un'entità

In questo momento il mercato offre molti strumenti per l'amministrazione di database, sia per i database di tipo relazionale (phpMyAdmin per MySQL <http://www.phpmyadmin.net>, pgAdmin per PostgreSQL <http://www.pgadmin.org/>), sia per database di tipo NoSQL (<http://docs.mongodb.org/ecosystem/tools/administration-interfaces/>). Solitamente l'utente finale di questi strumenti è un esperto in database administration (DBA) o uno sviluppatore *software*. In generale, quindi, sono strumenti sviluppati per personale con alte competenze tecniche.

L'amministrazione dei dati di business presenta problematiche diverse rispetto all'amministrazione dei database. Nel primo caso, infatti, i dati vengono aggregati sulla base di regole che variano a seconda del dominio applicativo. Inoltre, l'utente finale in questo caso non è sempre, e neppure deve essere esperto di informatica ma piuttosto del dominio di business.

Per risolvere questa problematica il mercato rende disponibili per alcuni *stack* tecnologici¹, che utilizzando linguaggi di alto livello, permettono allo sviluppatore di creare rapidamente pagine web utili all'amministrazione dei dati di *business*. Questo approccio prende spunto da una metodologia di gestione dei progetti di tipo *Agile* (<http://agilemanifesto.org/>).

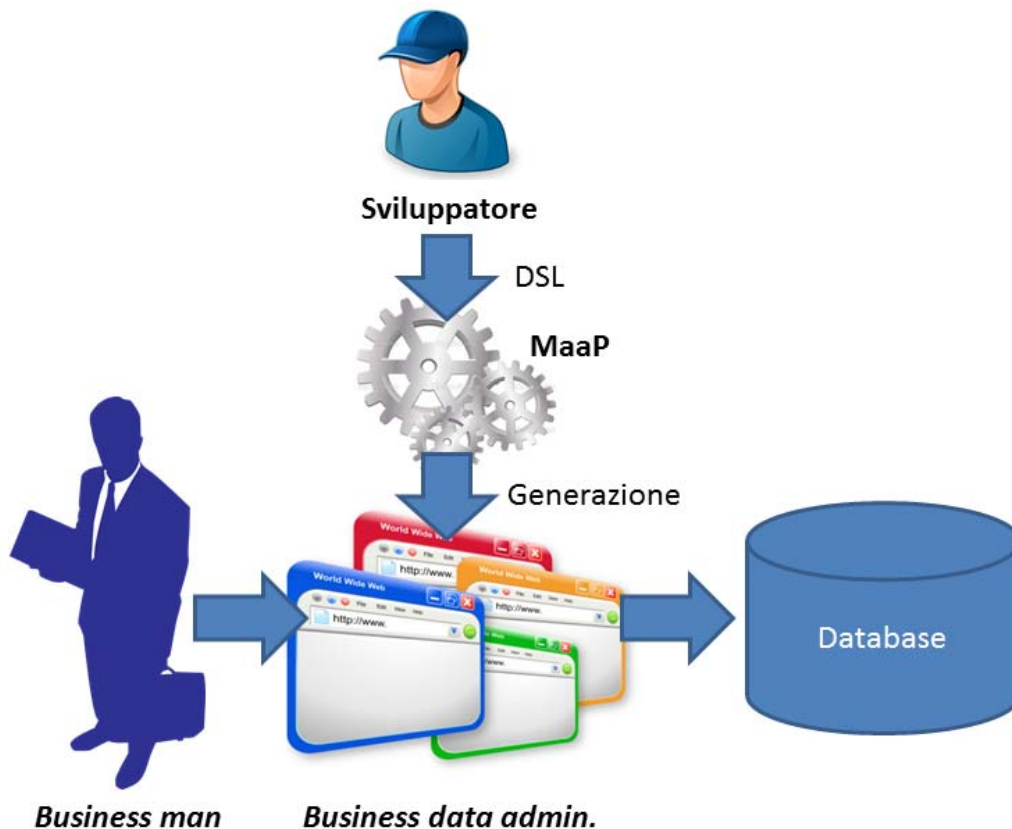
In questo modo, appoggiandosi sulla produttività dell'ambiente di programmazione, lo sviluppatore può rispondere in modo rapido e *standard* alle richieste delle esperti di *business*, fornendo loro degli strumenti adeguati. Ad esempio, per lo *stack* Ruby on Rails, è disponibile attualmente lo strumento Active Admin (<http://www.activeadmin.info/>, demo: <https://github.com/gregbell/demo.activeadmin.info>).

Business Data Administration su *stack* Node.js e MongoDB

Scopo del progetto è produrre un *framework* per generare interfacce web di amministrazione dei dati di *business* basati sullo *stack* Node.js e MongoDB: **MaaP** (MongoDB as an admin Platform). Il fruitore finale delle pagine generate è infatti l'esperto di *business*. Il prodotto atteso è fortemente ispirato ad Active Admin nel mondo Ruby on Rails.

MaaP è un framework che permette la creazione di pagine web di visualizzazione di dati provenienti da MongoDB (*collections* e *documents*) in maniera semplice e veloce. MongoDB è un sistema gestionale di basi di dati non relazionali, orientato ai documenti, di tipo NoSQL. Il linguaggio utilizzato per la gestione dei dati è JavaScript.

¹ Spring Roo (<http://projects.spring.io/spring-roo/>) per Java, Griffon (<http://griffon.codehaus.org/>) per Groovy, Play (<http://www.playframework.com/>) per Scala e Java



Lo sviluppatore, utilizzando MaaP come *framework* di lavoro, deve essere in grado di produrre due tipi di pagine:

1. *Collection-index*
2. *Collection-show*

Per ognuno di questi tipi di pagina il *framework* riceve in ingresso dallo sviluppatore gli opportuni dati: ad esempio, per le pagine di tipo *Collection*, riceve la collezione, gli attributi e gli elementi da visualizzare. Successivamente viene generata la pagina corrispondente.

Una pagina generata senza parametri deve utilizzare dei valori predefiniti. Il comportamento di *default* della pagina *Collection*, per esempio, è quello di visualizzare tutti i documenti con tutti gli attributi per un massimo prestabilito di documenti per pagina.

Di seguito si riportano degli esempi di utilizzo del *framework*. La sintassi degli esempi e le interfacce di esempio sono ricavate direttamente da Active Admin. Per quanto riguarda le interfacce è atteso ispirarsi ad Active Admin per funzionalità e grafica. Il linguaggio descrittivo (DSL, *Domain Specific Language*) utilizzato negli esempi tuttavia è da considerarsi solo a scopo dimostrativo. Il DSL da produrre sarà necessariamente ispirato al mondo Javascript. Non sono comunque posti vincoli stringenti al riguardo.

Per registrare con parametri di *default* una *Collection*, ad esempio accessibile attraverso l'URL `/admin/users`, si può ipotizzare per esempio il seguente comando:

```
MongoAdmin.register User, :namespace => "admin"
```

Il comando genera la pagina "*index*" e la pagina "*show*" relative alla *Collection*.

All'interno della pagina "*index*" (vedi figura 1) è riportata la visualizzazione in forma tabellare dell'elenco di tutti i documenti della *Collection* MongoDB "*user*". Ogni riga della tabella deve visualizzare tutte le chiavi del documento. La tabella deve essere paginata, ossia visualizzare un numero prestabilito di righe/documenti per pagina. Ogni riga della tabella deve avere una chiave selezionabile (ad esempio, la chiave "*id*") che rimanda alla corrispondente pagina "*show*" (ad esempio, `/admin/users/:id`, si veda la Figura 2).

The screenshot shows the Active Admin Depot interface for the 'Customers' collection. The main content area displays a table of customer records with columns for Id, Username, Email, and Created At. Each row includes a checkbox and links for View, Edit, and Delete. A filters sidebar on the right allows searching by Username, Email, and Created At. The table contains 20 records, all created on October 18, 2013.

Id	Username	Email	Created At
101	nigel99	theresa.doyle99@purdy.biz	October 18, 2013 01:21
100	adam.pagac98	briana_goyette98@jerde.net	October 18, 2013 01:21
99	marcel97	nora.klocko97@hermann.net	October 18, 2013 01:21
98	isobel_balistreri96	florian96@rolfson.info	October 18, 2013 01:21
97	trisha95	alysa95@hirthedubuque.org	October 18, 2013 01:21
96	stacy.jacobs94	montana94@stammritchie.com	October 18, 2013 01:21
95	isaias93	roel_reynolds93@ratke.biz	October 18, 2013 01:21
94	susana92	cody_rath92@schaden.net	October 18, 2013 01:21
93	dock91	vince91@graham.net	October 18, 2013 01:21
92	sadye.schimmel90	toy90@lakin.biz	October 18, 2013 01:21
91	ansley89	anna_baumbach89@braun.com	October 18, 2013 01:21
90	nya_champlin88	ignacio.pollich88@gaylordnitzsche.name	October 18, 2013 01:21
89	sarah87	anita_ward87@wuckert.info	October 18, 2013 01:21
88	madaline.beer86	nils86@jenkins.biz	October 18, 2013 01:21
87	ana85	candice_jacobi85@ankunding.com	October 18, 2013 01:21
86	candace.ondricka84	merle84@baileyrohan.org	October 18, 2013 01:21
85	demond83	theron_nolan83@konopelski.biz	October 18, 2013 01:21
84	keely_lemke82	tatum82@paucek.biz	October 18, 2013 01:21

Figura 1. Esempio di pagina *Collection-index*

La pagina predefinita “*show*”, deve visualizzare in forma estesa tutte le coppie chiavi valore.

Una volta registrata la pagina *Collection* questa deve creare in automatico una voce nel menu di accesso delle varie *Collections*. Deve poter essere configurabile posizione e nome della voce nel menu.

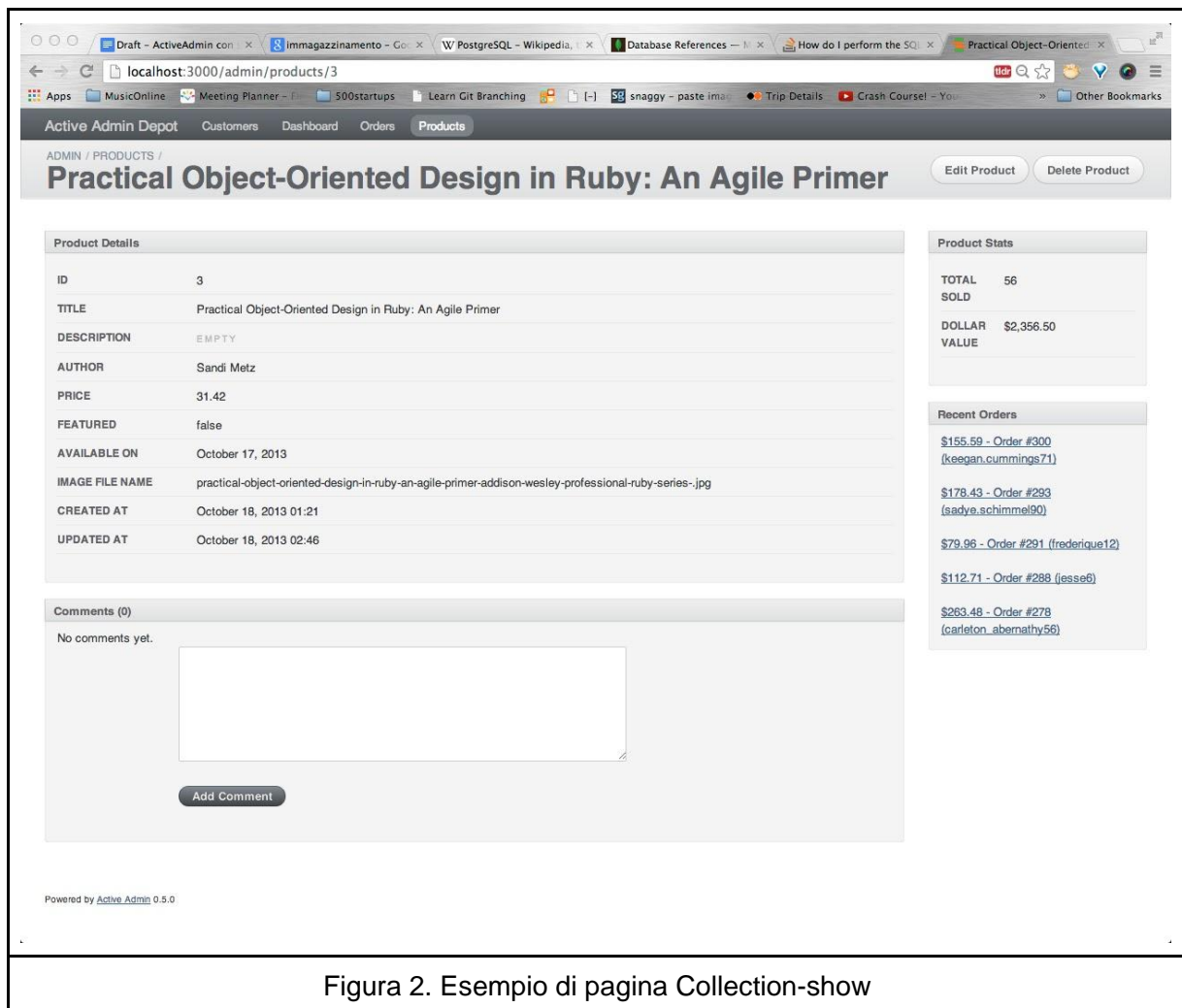


Figura 2. Esempio di pagina *Collection-show*

Nel caso l'utente abbia permessi di scrittura i campi del documento devono poter essere editabili.

Non è richiesta la creazione di nuovi documenti o la modifica di un insieme di documenti. L'analisi di queste caratteristiche è complessa, ma interessante poiché all'interno di MongoDB sono presenti un numero limitato di vincoli sui tipi di dati e le loro relazioni. Permettere la creazione di documenti potrebbe corrompere il funzionamento delle applicazioni che fanno riferimento al database.

Una parte opzionale interessante riguarda la possibilità di creare nuovi documenti mantenendo i vincoli definiti dai modelli. Poiché MongoDB non fornisce *join* tra documenti, questa funzionalità (*populate*) è fornita necessariamente da librerie di terze parti. Una di queste è Mongoose (<http://mongoosejs.com/>). Questo aggiunge una serie di difficoltà in quanto non è chiaro come importare questi modelli all'interno di MaaP in maniera semplice (ricordando che lo scopo ultimo di MaaP è di creare interfacce di amministrazione dati in maniera veloce).

Requisiti minimi

Di seguito vengono elencati i requisiti che il prodotto deve obbligatoriamente soddisfare:

1. Realizzazione di MaaP, per la generazione rapida di *client web* per la visualizzazione di dati provenienti da MongoDB. Si richiede l'implementazione delle funzionalità relative alle pagine *Collection show* e *index*.
2. Lo *stack* tecnologico richiesto include:
 - a. [Node.js](http://nodejs.org/) (<http://nodejs.org/>) per la realizzazione della componente server;
 - b. [Express](http://expressjs.com/) (<http://expressjs.com/>) per la realizzazione dell'infrastruttura della *web application* generata;
 - c. [Mongoose.js](http://mongoosejs.com/) (<http://mongoosejs.com/>) per l'interfacciamento con il database;
 - d. [MongoDB](#) per il recupero dei dati.

Non è posto alcun vincolo circa la scelta dei *framework* riguardanti la realizzazione della componente di *front-end* (i.e. [Angular.js](#), [Ember.js](#), etc etc).

3. Definizione di un linguaggio astratto, DSL (*Domain Specific Language*) da utilizzare per la generazione delle pagine. Il linguaggio dovrà essere testuale. Non è necessario fornire un editor specializzato.
4. L'accesso alle pagine generate deve essere mediato da un sistema di autenticazione basato su *email/password* con possibilità di recupero *password*. Il sistema di autenticazione recupera e persiste i dati in un database indipendente da quello in analisi, in maniera simile al progetto open-source (<https://github.com/jedireza/drywall/>). Devono essere presenti almeno due categorie (profili) di utenti:
 - a. "user": permessi di sola lettura e visualizzazione del database in analisi
 - b. "admin":
 - i. potere di creazione/rimozione utenti
 - ii. potere di elevare/declassare utenti a livello di "admin"
 - iii. permessi di scrittura nel database in analisi (modifica dati o creazioni indici)
5. Possibilità di effettuare il *deployment* su [Heroku](https://www.heroku.com/) (<https://www.heroku.com/>, servizio *cloud* di tipo *platform as a service*). Tramite Heroku è possibile rendere disponibile *on-line* applicazioni *web* scritte in vari framework (Node.js, Ruby on Rails e Play alcuni di essi). Heroku si rende responsabile di ospitare il codice (con relative dipendenze/librerie) adottando un meccanismo di deployment basato su git (<http://git-scm.com/>, software di versionamento distribuito).
6. Pubblicazione del progetto su GitHub e utilizzo delle *issue* di Github per segnalare *bug*.
7. Compatibilità con la versione 30.0.x o superiori di Chrome e la versione 24.x o superiori di Firefox.

Requisiti opzionali

Ramo servizio: MongoDB as an admin Service (MaaS).

1. Adattare il *framework* sviluppato per offrire il prodotto come servizio web. In maniera similare a quanto offerto da Automattic (<http://automattic.com/>) con Wordpress o Ghost (<http://ghost.org/>, sviluppato utilizzando lo stesso *stack* tecnologico richiesto dal presente progetto).
2. L'utente di MaaS essere in grado di scrivere le proprie pagine direttamente da web con una di queste due modalita': 1) tramite un editor di testo; 2) caricando un file prodotto grazie al prodotto open-source MaaP. L'architettura deve tener conto di poter accogliere queste modifiche, in particolare le pagine definite dagli utenti non devono esser salvate su disco, ma direttamente in database.
3. Una volta definite le proprie pagine l'utente deve essere in grado di usufruire delle stesse direttamente da MaaS utilizzando un collegamento costruito ad-hoc. Es: L'utente Rossi crea una applicazione "AI". Crea la pagina "Vincoli che interroga la *collection* corrispondente. Una volta creata la pagina vuole distribuire la sua applicazione "AI" ad un numero ristretto di utenti privati. La sua applicazione sara' quindi disponibile all'indirizzo: <http://maas.com/users/frossi/vincoli>. Solo gli utenti autorizzati potranno accedervi e usufruire dei contenuti.

Ramo linguaggio.

4. Possibilità di definire pulsanti. Un pulsante risponde eseguendo del codice *custom* (Javascript) usando come contesto il *Document* o la *Collection* in cui è richiamato (i.e. nel caso *Collection*, l'insieme di documenti deve essere visibile al codice eseguito).
5. Proposta proattiva del sistema di creazione di indici per il database, sulla base delle *query* più richieste attraverso il *framework*. Nel caso di utenti con permessi di scrittura ("admin") deve essere resa disponibile la funzionalità di creazione di tali indici.
6. Creazione di nuovi *Document* all'interno della base dati, rispettando la *business logic* dell'applicazione generata.

Variazione dei requisiti

Non sono ammesse variazioni se non a evidente miglioramento di quanto richiesto dal committente. Non è esclusa la comunicazione, da parte del committente, di variazioni ai requisiti sia precedentemente alla consegna delle offerte che durante la realizzazione del sistema.

Documentazione

La consegna del sistema dovrà essere accompagnata dai necessari manuali d'uso e da ogni altra documentazione tecnica necessaria per l'utilizzo del prodotto da parte del personale operatore del committente. È gradita la versione bilingue italiano e inglese.

Garanzia e manutenzione

Il fornitore dovrà garantire in sede di collaudo (Revisione di Accettazione) il funzionamento corretto del sistema. L'eliminazione dei difetti e delle non conformità eventualmente emersi in sede di collaudo sono a totale carico del fornitore.

Le modalità di collaudo saranno proposte dal fornitore e costituiranno titolo per la valutazione dell'offerta ai fini dell'aggiudicazione dell'appalto. I dati di collaudo costituiscono parte integrante delle modalità di collaudo. Le modalità di collaudo saranno considerate definitive e contrattualmente vincolanti solo a seguito di formale approvazione da parte del committente.

Rinvio

Per tutto quanto non previsto nel presente capitolato, sono applicabili le disposizioni contenute nelle leggi e nei collegati per la gestione degli appalti pubblici.

Proponente

CoffeeStrap (<http://www.coffeestrapp.com>) è un *language services marketplace* che offre a chiunque desideri insegnare la propria lingua ad apprendere un linguaggio straniero la possibilità di entrare in contatto con studenti e insegnanti attraverso *online* e *offline workshops*, che costituiscano sessioni di apprendimento sulla base degli interessi dell'utente.

L'azienda è incorporata negli Stati Uniti ed ha attualmente base strategica nell'ecosistema *startup* di San Francisco e della Bay Area.

CoffeeStrap è rappresentato da:

Mahesh Casiraghi: PhD Cognitive Neuroscience, University of Padova, 2013.

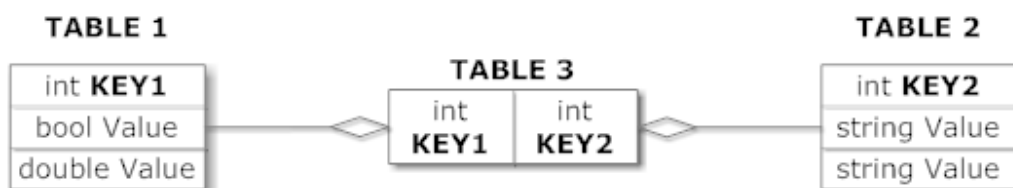
Alessandro Maccagnan (contatto di riferimento per MaaP): PhD Computer Science, University of Padova, 2012, am@coffeestrapp.com.

Appendice

MongoDB

Un'istanza di database MongoDB composta è da un insieme di *Collection*. Una *Collection* è composta da un insieme di *Document*. Un *Document* è un insieme di coppie chiave-valore. I *Document* possono avere uno schema dinamico, ovvero le strutture dei documenti di una singola collezione possono differire tra di loro e chiavi comuni a documenti in una collezione possono avere diversi tipi di dati. Una differenza fondamentale rispetto ad un database relazionale è l'assenza di vincoli di integrità referenziale.

Relational Model



Document Model

Collection ("Things")



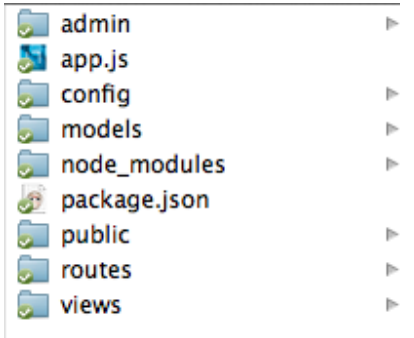
Per ovviare a questa problematica, nello stack Node.js si utilizza un *Object Document Mapper* (ODM), Mongoose.js (<http://mongoosejs.com/>), che permette la definizione di alcuni vincoli e schemi. La logica di una applicazione è perciò definita in Node.js tramite l'utilizzo di schemi e modelli Mongoose.

MaaP

L'utilizzo del *framework* da parte dello sviluppatore si realizza in due distinte azioni.

Azione 1

La azione prevede la creazione da linea di comando dello scheletro del progetto.

Input	Output
From console: \$ MongoAdmin --initialize newproj && cd newproj && npm install	

Dettagli Implementativi

Lo scheletro del progetto generato deve comprendere almeno le seguenti componenti:

- librerie necessarie al funzionamento del progetto (i.e. librerie “classiche”: mongoose/express/etc etc, librerie proprietarie (necessarie a MaaP)
- file di configurazione necessari al funzionamento dell'applicazione nella loro versione predefinita (i.e. database per sistema di autenticazione, mail server, etc etc)
- sistema di autenticazione (email/password/recupero password)
- directory di descrizione pagine web.

Una volta creato lo scheletro del progetto sarà possibile eseguire l'applicazione e vedere il sistema di autenticazione con un menu vuoto.

Azione 2

La seconda fase prevede la definizione delle pagine web da visualizzare sulla base del linguaggio DSL definito. Il *framework* deve essere in grado di interpretare i file di descrizione e generare dinamicamente le richieste al database da analizzare e le pagine web di visualizzazione.

Input	Output																					
<pre>MongoAdmin.register User do index do column :id column :email column :name column :age column :sex end show do ad attributes_table do row :email row :name row :chat_name row :sex row :location row :sign_in_count row :provider row :authentication_token row :registration_ip row :current_sign_in_ip row :last_sign_in_ip end end end</pre>	<table border="1"><tbody><tr><td>366</td><td>giulio.gavardi@gmail.com</td><td>Giulio Gavardi</td><td>30</td><td>Male</td><td>Padova, Veneto, Italy</td><td>C</td></tr><tr><td>351</td><td>927d821dd8f77f3a69@coffeestrap.com</td><td></td><td>43</td><td>Male</td><td>Padova, Veneto, Italy</td><td>C</td></tr><tr><td>329</td><td>riccardo.cardin@gmail.com</td><td>Riccardo Cardin</td><td>30</td><td>Male</td><td>Padova, Veneto, Italy</td><td>C</td></tr></tbody></table>	366	giulio.gavardi@gmail.com	Giulio Gavardi	30	Male	Padova, Veneto, Italy	C	351	927d821dd8f77f3a69@coffeestrap.com		43	Male	Padova, Veneto, Italy	C	329	riccardo.cardin@gmail.com	Riccardo Cardin	30	Male	Padova, Veneto, Italy	C
366	giulio.gavardi@gmail.com	Giulio Gavardi	30	Male	Padova, Veneto, Italy	C																
351	927d821dd8f77f3a69@coffeestrap.com		43	Male	Padova, Veneto, Italy	C																
329	riccardo.cardin@gmail.com	Riccardo Cardin	30	Male	Padova, Veneto, Italy	C																

User Details	
EMAIL	alessandro.maccagn
NAME	Alessandro Maccagn
CHAT NAME	alemhnan
AGE	EMPTY
SEX	EMPTY
LOCATION	EMPTY
SIGN IN COUNT	8444
PROVIDER	facebook
PROVIDER UID	686209457
AUTHENTICATION TOKEN	cNgpyVW4pAtqyfY4A
REGISTRATION IP	EMPTY
CURRENT SIGN IN IP	72.3.188.246
LAST SIGN IN IP	72.3.188.246
LEARNINGS	EMPTY
KNOWINGS	EMPTY

Di seguito si presentano degli esempi di sintassi necessari alla creazione delle pagine di tipo *Collection-index* e *Collection-show*.

Collection-index

All'interno delle pagine *Collection-index* deve essere possibile definire

- una serie di attributi da visualizzare
- un ordinamento
- un eventuale limite di elementi da visualizzare

Opzioni di configurazione "index"

La pagina “index” deve poter essere configurata indicando quali chiavi visualizzare all’interno della tabella.

Es: (collection “user”):

```
index do
  column "id", :_id
  column "Data di creazione", :modifiedAt
  column "Email", :email
  column "username", :profile.username
end
```

Tale sintassi permette la visualizzazione di una tabella con i soli gli attributi “user_id”, “user.modifiedAt”, “user.profile.username”. Ogni colonna deve poter accettare un valore da visualizzare. Tale valore puo’ essere un campo del documento attuale oppure una sua trasformazione. Una trasformazione puo’ essere definita da una funzione Javascript fornita dall’utente e con input sottinteso il documento attuale e output definito dal codice della funzione.

Es:

```
index do
  column "Data di creazione", :modifiedAt.toISOString()
end
```

Ipotizzando esistente e visibile la funzione “myDateFormat” :

```
index do
  column "Data di creazione", myDateFormat(:modifiedAt)
end
```

In caso di assenza di specifica, la tabella deve poter esser ordinata per data creazione dell’oggetto (specificata nella chiave “_id”: <http://docs.mongodb.org/manual/reference/object-id/>). Oltre a quello di *default*, devono poter essere definibili altri ordinamenti, indicando quali colonne devono essere ordinabili e secondo quale chiave.

Es:

```
index do
  column "Numero di accessi", :profile.loginCount., :sortable =>true,
:sortable => profile.lastLogin
end
```

In caso di chiavi che facciano riferimento a documenti esterni deve poter essere definibile quali di queste chiavi popolare in automatico (vedi [“populate” in mongoosejs](#)). La funzionalità “populate” di Mongoose è simile per concetto alle operazioni di *join* in un database relazionale.

Un documento puo' far riferimento al suo interno ad altri documenti. Il documento "utente" puo' contenere un riferimento ad un documento "profilo". La funzionalità "populate" quindi si riferisce ad un modo di interrogare il database recuperando il documento "utente" con automaticamente innestato il documento figlio "profilo".

Es:

```
index populate :contacts do
  column "Numero di contatti", :contacts.length
  column "Nome contatti", :contacts.name
end
```

Infine, devono poter essere definite delle *query* da utilizzare per creare la pagina risultante sulla base del risultato della loro estrazione.

Es:

```
mongoDBQuery = {'status.discarded': { $ne: true }}

index :query => mongoDBQuery do
  column "id", :_id
  column "email", :email
  column "status", :status
end
```

in cui `mongoDBQuery` e' una *query* propria di MongoDB fornita tramite la libreria Mongoose.

Collection-show

All'interno delle pagine Collection-show deve essere possibile definire:

- una serie di attributi da visualizzare
- un popolamento di sotto attributi

Opzioni di configurazione "show"

La pagina "index" deve poter essere configurata indicando quali chiavi visualizzare al suo interno.

Es:

```
show do
  row :id
  row :email
  row "username", :profile.username
end
```

In caso di chiavi che facciano riferimento a documenti esterni deve poter essere definibile quali di queste chiavi popolare in automatico (vedi [“populate” in mongoosejs](#)).

Es:

```
show populate :cities do
  column "Numero di citta' visitate", :cities.length
  column "Elenco citta'", :cities.name
end
```