MaaP: MongoDB as an admin Platform

Definizione di Prodotto

Versione	1.2.0
Data creazione	2014-04-21
Data ultima modifica	2014-05-19
Stato del Documento	Formale
Uso del Documento	Esterno
Redazione	Pinato Giacomo, Alberto Garbui
	Alessandro Benetti, Andrea Perin
Verifica	Fabio Miotto, Michele Maso
Approvazione	Mattia Sorgato
Distribuzione	Aperture Software
	Prof. Vardanega Tullio
	Prof. Cardin Riccardo
	CoffeeStrap

Sommario

Architettura di dettaglio dell'applicazione MaaP: MongoDB as an admin Platform.

Diario delle modifiche

Versione	Data	Autore	Modifiche effettuate
1.2.0	2014-05-19	Mattia Sorgato (RE)	Approvazione documento.
1.1.1	2014-05-19	Fabio Miotto (VR)	Verifica documento.
1.1.0	2014-05-16	Michele Maso (VR)	Verifica documento.
1.0.4	2014-05-08	Alessandro Benetti (PR)	Stesura Datamanager.
1.0.3	2014-05-01	Alberto Garbui (PR)	Stesura controller del Server
1.0.2	2014-04-29	Andrea Perin (PR)	Stesura dei controller
1.0.2	2014-04-23	Giacomo Pinato (PR)	Stesura dei servizi e dei controller del Client
1.0.1	2014-04-21	Alessandro Benetti (PR)	Prima stesura del documento.

Tabella 1: Registro delle modifiche

Indice

1 Introduzione **6**
1.1 Scopo del documento . . . 6
1.2 Scopo del prodotto . . . 6
1.3 Glossario . . . 6
1.4 Riferimenti . . . 6
1.4.1 Normativi . . . 6
1.4.2 Informativi . . . 6
2 Standard di progetto **7**
2.1 Standard di progettazione architetturale . . . 7
2.2 Standard di documentazione del codice . . . 7
2.3 Standard di programmazione . . . 7
2.3.1 Tipi in JavaScript . . . 7
2.4 Strumenti di lavoro . . . 7
3 Namespace MaaP **8**
4 Specifica componenti MaaP::Server **8**
4.1 ::Controller . . . 8
4.1.1 @dispatcher . . . 8
4.1.2 @frontController . . . 16
4.1.3 @index . . . 17
4.1.4 @passport . . . 17
4.2 ::ModelServer . . . 18
4.2.1 ::Database . . . 18
4.2.1.1 @index . . . 18
4.2.1.2 @MongooseDBAnalysis . . . 19
4.2.1.3 @MongooseDBFramework . . . 20
4.2.2 ::DataManager . . . 20
4.2.2.1 ::DatabaseAnalysisManager . . . 20
4.2.2.1.1 @DatabaseAnalysisManager . . . 20
4.2.2.1.2 @DataRetrieverAnalysis . . . 22
4.2.2.2 ::DatabaseUserManager . . . 26
4.2.2.2.1 @DatabaseUserManager . . . 26
4.2.2.2.2 @DataRetrieverUsers . . . 29
4.2.2.3 ::IndexManager . . . 31
4.2.2.3.1 @IndexManager . . . 31
4.2.2.4 @JsonComposer . . . 33
4.2.3 ::DSL . . . 36
4.2.3.0.1 @DSLManager . . . 36
4.2.3.0.2 @DSLParser . . . 37
4.2.3.0.3 @index . . . 38
4.2.3.0.4 @JavascriptParser . . . 39
4.2.3.0.5 @schemaGenerator . . . 39
5 Specifica componenti Maap::Client **40**
5.1 ::View . . . 40
5.1.1 ::Template . . . 40
5.2 ::ControllerModelView . . . 40
5.2.1 ::ControllerClient . . . 40
5.2.1.1 @CollectionController . . . 40

5.2.1.2 @DashboardController . . . 42
5.2.1.3 @DocumentController . . . 43
5.2.1.4 @DocumentEditController . . . 44
5.2.1.5 @IndexController . . . 45
5.2.1.6 @LoginController . . . 46
5.2.1.7 @NavBarController . . . 47
5.2.1.8 @ProfileController . . . 48
5.2.1.9 @ProfileEditController . . . 48
5.2.1.10 @QueryController . . . 49
5.2.1.11 @RegisterController . . . 51
5.2.1.12 @UsersCollectionController . . . 52
5.2.1.13 @UsersController . . . 54
5.2.1.14 @UsersEditController . . . 55
5.2.2 ::Directives . . . 56
5.2.2.1 @Passwrod_check . . . 56
5.2.2.2 @Unique . . . 56
5.3 ::ModelClient . . . 56
5.3.1 ::Services . . . 56
5.3.1.1 @AuthService . . . 56
5.3.1.2 @CollectionDataService . . . 57
5.3.1.3 @CollectionListService . . . 57
5.3.1.4 @DocumentDataService . . . 57
5.3.1.5 @DocumentEditservice . . . 58
5.3.1.6 @IndexService . . . 58
5.3.1.7 @LogoutService . . . 59
5.3.1.8 @ProfileDataService . . . 59
5.3.1.9 @ProfileEditService . . . 59
5.3.1.10 @QueryService . . . 60
5.3.1.11 @RegisterService . . . 60
5.3.1.12 @UserCollectionService . . . 61
5.3.1.13 @UserDataService . . . 61
5.3.1.14 @UserEditService . . . 61
6 Diagrammi di sequenza 63
6.1 Modifica della View con successo . . . 63
6.2 Modifica della View con insuccesso . . . 64

Elenco delle figure

1 Diagramma sequenza: Modifica View con successo 63
2 Diagramma sequenza: Modifica View con insuccesso 64

1 Introduzione

1.1 Scopo del documento

Il seguente documento ha lo scopo di definire nel dettaglio la struttura del sistema *MaaP*$_G$, approfondendo quanto già riportato nella Specifica Tecnica. Tale documento fornisce una struttura dettagliata e completa che viene utilizzata dai *Programmatori* per le attività di codifica.

1.2 Scopo del prodotto

Lo scopo del prodotto è produrre un *framework*$_G$ per generare interfacce web di amministrazione dei dati di business basato sullo stack *Node.js*$_G$ e *MongoDB*$_G$.
L'obiettivo è quello di semplificare il lavoro allo *sviluppatore*$_G$ che dovrà rispondere in modo rapido e standard alle richieste degli esperti di *business*$_G$.

1.3 Glossario

Al fine di evitare ogni ambiguità nella comprensione del linguaggio utilizzato nel presente documento e, in generale, nella documentazione fornita dal gruppo Aperture Software, ogni termine tecnico, di difficile comprensione o di necessario approfondimento verrà inserito nel documento *Glossario_v4.2.0.pdf*.
Saranno in esso definiti e descritti tutti i termini in corsivo e allo stesso tempo marcati da una lettera "G" maiuscola in pedice nella documentazione fornita.

1.4 Riferimenti

1.4.1 Normativi

- **Analisi dei requisiti**: *Analisi_dei_Requisiti_v4.2.0.pdf*;
- **Norme di Progetto**: *Norme_di_Progetto_v4.2.0.pdf*;
- **Specifica Tecnica**: *Specifica_Tecnica_v4.2.0.pdf*;

1.4.2 Informativi

- **AngularJS API**: https://docs.angularjs.org/api;
- **MongooseJS API**: http://mongoosejs.com/docs/api.html;
- **MongoDB API per Node.js**: http://mongodb.github.io/node-mongodb-native/;
- **Node.js API**: http://nodejs.org/api/.

2 Standard di progetto

2.1 Standard di progettazione architetturale

Gli standard di progettazione architetturale sono definiti nella *Specifica_Tecnica_v2.2.0.pdf*.

2.2 Standard di documentazione del codice

Gli standard di documentazione del codice sono specificati nel documento di *Norme_di_Progetto_v4.2.0.pdf*, nella sezione 4.3.5.1.2.

2.3 Standard di programmazione

Gli standard di programmazione sono specificati nel documento di *Norme_di_Progetto_v4.2.0.pdf*, nella sezione 4.3.5.

2.3.1 Tipi in JavaScript

Essendo JavaScript un linguaggio non tipato, una variabile può assumere diversi tipi durante l'esecuzione di un codice. Infatti è possibile cambiare il tipo di un riferimento semplicemente assegnando un valore diverso ad una data variabile. È quindi ambiguo definire all'interno di un frammento di codice il tipo statico che una variabile possiede. Lo stesso vale per i valori ritornati da una funzione JavaScript, i quali possono essere addirittura funzioni.

2.4 Strumenti di lavoro

Gli strumenti di lavoro sono trattati nelle sezioni 4.3.5.2.2 e 4.3.5.3.2 del documento di *Norme_di_Progetto_v4.2.0.pdf*.

3 Namespace MaaP

Namespace generale del progetto. In accordo con il design pattern architetturale Client-Server che abbiamo adottato, le interazioni che il Client ha con il Server sono di tipo REST-like.

4 Specifica componenti MaaP::Server

Package Server del Design Pattern Client-Server.

4.1 ::Controller

Package che gestisce il dialogo con il Client di MaaP.

4.1.1 @dispatcher

Descrizione

Modulo di inizializzazione della componente Server utilizzata per la trasmissione dei dati al Client.
Il modulo *dispatcher* gestisce tutte le richieste effettuate dal Client al Server, effettuando le dovute redirezioni e le chiamate per ottenere i dati dai database nel Server. Le richieste che pervengono al *dispatcher* sono di tipo REST, a cui il modulo stesso risponde con il JSON della risorsa richiesta. Questo avviene solamente se l'utente che ha richiesto la risorsa è registrato ed autenticato, prima di effettuare la richiesta, e che abbia i dovuti permessi per effettuare la richiesta. Infatti, prima di eseguire ogni richiesta di accesso ai dati del Server, il *dispatcher* esegue una chiamata al modulo *passport* per verificare l'autenticazione del richiedente e per distinguere gli utenti standard dagli utenti amministratori.
Per ultima cosa, il *dispatcher* gestisce eventuali richieste di risorse inesistenti o errate reindirizzando il chiamante alla pagina iniziale del Client AngularJS.

Dipendendenze

- @passport;
- path;
- @DatabaseAnalysisManager;
- @DatabaseUserManager;
- @IndexManager.

Metodi

Gestione Collection e Document

get('/api/collection/list', passport.checkAuthenticated, datamanager.sendCollectionsList)

Parametri

'/api/collection/list'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticated
Funzione di *@passport* che verifica che l'utente sia autenticato.

datamanager.sendCollectionsList
Callback da richiamare al termine dell'autenticazione.

Descrizione

Intercetta la richiesta di generare l'elenco delle Collection per visualizzarlo, verificando che essa sia autenticata e richiama il metodo *sendCollectionsList* del *@datamanager*.

get('/api/collection/:col_id', passport.checkAuthenticated, datamanager.sendCollection)

Parametri

'/api/collection/:col_id'

Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticated

Funzione di *@passport* che verifica che l'utente sia autenticato.

datamanager.sendCollection

Callback da richiamare al termine dell'autenticazione.

Descrizione

Intercetta la richiesta di generare l'elenco dei Document all'interno di una Collection specificata con *:col_id*. Verifica che essa sia autenticata e richiama il metodo *sendDocument* del *@datamanager*.

get('/api/collection/:col_id/:doc_id', passport.checkAuthenticated, datamanager.sendDocument)

Parametri

'/api/collection/:col_id/:doc_id'

Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticated

Funzione di *@passport* che verifica che l'utente sia autenticato.

datamanager.sendDocument

Callback da richiamare al termine dell'autenticazione.

Descrizione

Intercetta la richiesta di generare i dati di un Document specificato con *:doc_id* all'interno della Collection specificata con :col_id per visualizzarli. Verifica che essa sia autenticata e richiama il metodo *sendDocument* del *@datamanager*.

get('/api/collection/:col_id/:doc_id/edit', passport.checkAuthenticatedAdmin, datamanager.sendDocumentEdit)

Parametri

'/api/collection/:col_id/:doc_id/edit'

Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticatedAdmin

Funzione di *@passport* che verifica che l'utente sia autenticato e che il suo livello di accesso sia Amministratore.

datamanager.sendDocumentEdit

Callback da richiamare al termine dell'autenticazione.

Descrizione

Intercetta la richiesta di generare i dati di un Document specificato con *:doc_id* all'interno della Collection specificata con :col_id per modificarli. Verifica che essa sia autenticata e richiama il metodo *sendDocumentEdit* del *@datamanager*.

put('/api/collection/:col_id/:doc_id/edit', passport.checkAuthenticatedAdmin, datamanager.updateDocument)

Parametri

'/api/collection/:col_id/:doc_id/edit'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticatedAdmin
Funzione di *@passport* che verifica che l'utente sia autenticato e che il suo livello di accesso sia Amministratore.

datamanager.updateDocument
Callback da richiamare al termine dell'autenticazione.

Descrizione

Intercetta la richiesta di modificare i dati di un Document specificato con *:doc_id* all'interno della Collection specificata con :col_id. Verifica che essa sia autenticata come amministratore e richiama il metodo *updateDocument* del *@datamanager*.

delete('/api/collection/:col_id/:doc_id/edit', passport.checkAuthenticatedAdmin, datamanager.removeDocument)

Parametri

'/api/collection/:col_id/:doc_id/edit'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticatedAdmin
Funzione di *@passport* che verifica che l'utente sia autenticato e che il suo livello di accesso sia Amministratore.

datamanager.removeDocument
Callback da richiamare al termine dell'autenticazione.

Descrizione

Intercetta la richiesta di cancellare un Document specificato con *:doc_id* all'interno della Collection specificata con :col_id. Verifica che essa sia autenticata come amministratore e richiama il metodo *removeDocument* del *@datamanager*.

Gestione Profilo Utente

get('/api/profile', passport.checkAuthenticated, usermanager.sendUserProfile)

Parametri

'/api/profile'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticated
Funzione di *@passport* che verifica che l'utente sia autenticato.

usermanager.sendUserProfile
Callback da richiamare al termine dell'autenticazione.

Descrizione

Intercetta la richiesta di generare i dati del profilo utente per visualizzarli, verificando che essa sia autenticata e richiama il metodo *sendUserProfile* del *@usermanager*.

get('/api/profile/edit', passport.checkAuthenticated, usermanager.sendUserProfileEdit)

Parametri

'/api/profile/edit'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticated
Funzione di *@passport* che verifica che l'utente sia autenticato.

usermanager.sendUserProfileEdit
Callback da richiamare al termine dell'autenticazione.

Descrizione
Intercetta la richiesta di generare i dati del profilo utente per modificarli, verificando che essa sia autenticata e richiama il metodo *sendUserProfileEdit* del *@usermanager*.

put('/api/profile/edit', passport.checkAuthenticated, usermanager.updateUserProfile)

Parametri

'/api/profile/edit'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticated
Funzione di *@passport* che verifica che l'utente sia autenticato.

usermanager.updateUserProfile
Callback da richiamare al termine dell'autenticazione.

Descrizione
Intercetta la richiesta di modificare i dati del profilo utente, verificando che essa sia autenticata e richiama il metodo *updateUserProfile* del *@usermanager*.

Gestione Utenti

get('/api/users/list', passport.checkAuthenticatedAdmin, usermanager.getUsersList)

Parametri

'/api/users/list'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticatedAdmin
Funzione di *@passport* che verifica che l'utente sia autenticato e che il suo livello di accesso sia Amministratore.

usermanager.getUsersList
Callback da richiamare al termine dell'autenticazione.

Descrizione
Intercetta la richiesta di generare l'elenco degli Utenti per visualizzarlo, verificando che essa sia autenticata come ammistratore e richiama il metodo *getUsersListionsList* del *@usermanager*.

get('/api/users/:user_id', passport.checkAuthenticatedAdmin, usermanager.sendUser)

Parametri

'/api/users/:user_id'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticatedAdmin
Funzione di *@passport* che verifica che l'utente sia autenticato e che il suo livello di accesso sia Amministratore.

usermanager.sendUser
Callback da richiamare al termine dell'autenticazione.

Descrizione
Intercetta la richiesta di generare i dati di un Utente specificato con :user_id per visualizzarli. Verifica che essa sia autenticata come amministratore e richiama il metodo *sendUser* del *@usermanager*.

get('/api/users/:user_id/edit', passport.checkAuthenticatedAdmin, usermanager.sendUserEdit)

Parametri

'/api/users/:user_id/edit'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticatedAdmin
Funzione di *@passport* che verifica che l'utente sia autenticato e che il suo livello di accesso sia Amministratore.

usermanager.sendUserEdit
Callback da richiamare al termine dell'autenticazione.

Descrizione
Intercetta la richiesta di generare i dati di un Utente specificato per con :user_id per modificarli. Verifica che essa sia autenticata come amministratore e richiama il metodo *sendUserEdit* del *@usermanager*.

put('/api/users/:user_id/edit', passport.checkAuthenticatedAdmin, usermanager.updateUser)

Parametri

'/api/users/:user_id/edit'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticatedAdmin
Funzione di *@passport* che verifica che l'utente sia autenticato e che il suo livello di accesso sia Amministratore.

usermanager.updateUser
Callback da richiamare al termine dell'autenticazione.

Descrizione
Intercetta la richiesta di modificare i dati di un Utente specificato per con :user_id. Verifica che essa sia autenticata come amministratore e richiama il metodo *updateUser* del *@usermanager*.

delete('/api/users/:user_id/edit', passport.checkAuthenticatedAdmin, usermanager.removeUser)

Parametri

'/api/users/:user_id/edit'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticatedAdmin
Funzione di *@passport* che verifica che l'utente sia autenticato e che il suo livello di accesso sia Amministratore.

usermanager.removeUser
Callback da richiamare al termine dell'autenticazione.

Descrizione
Intercetta la richiesta di eliminare un Utente specificato per con :user_id.

Verifica che essa sia autenticata come amministratore e richiama il metodo *removeUser* del *@usermanager*.

Gestione Query più utilizzate

get('/api/queries/list', passport.checkAuthenticatedAdmin, datamanager.getTopQueries)

Parametri

'/api/queries/list'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticatedAdmin
Funzione di *@passport* che verifica che l'utente sia autenticato e che il suo livello di accesso sia Amministratore.

datamanager.getTopQueries
Callback da richiamare al termine dell'autenticazione.

Descrizione

Intercetta la richiesta di generare l'elenco delle query più utilizzate. Verifica che essa sia autenticata come amministratore e richiama il metodo *getTopQueries* del *@datamanager*.

delete('/api/queries/list', passport.checkAuthenticatedAdmin, datamanager.resetQueries)

Parametri

'/api/queries/list'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticatedAdmin
Funzione di *@passport* che verifica che l'utente sia autenticato e che il suo livello di accesso sia Amministratore.

datamanager.resetQueries
Callback da richiamare al termine dell'autenticazione.

Descrizione

Intercetta la richiesta di eliminare tutte le query più utilizzate. Verifica che essa sia autenticata come amministratore e richiama il metodo *resetQueries* del *@datamanager*.

Gestione Indici nel Database di analisi

get('/api/indexes', passport.checkAuthenticatedAdmin, datamanager.getIndexesList)

Parametri

'/api/indexes'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticatedAdmin
Funzione di *@passport* che verifica che l'utente sia autenticato e che il suo livello di accesso sia Amministratore.

datamanager.getIndexesList
Callback da richiamare al termine dell'autenticazione.

Descrizione

Intercetta la richiesta di generare l'elenco degli indici esistenti sul database di analisi. Verifica che essa sia autenticata come amministratore e richiama il metodo *getIndexesList* del *@datamanager*.

put('/api/indexes', passport.checkAuthenticatedAdmin, datamanager.createIndex)

Parametri

'/api/indexes'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticatedAdmin
Funzione di *@passport* che verifica che l'utente sia autenticato e che il suo livello di accesso sia Amministratore.

datamanager.createIndex
Callback da richiamare al termine dell'autenticazione.

Descrizione
Intercetta la richiesta di inserire un indice nel database di analisi. Verifica che essa sia autenticata come amministratore e richiama il metodo *createIndex* del *@datamanager*.

delete('/api/indexes/:index_name', passport.checkAuthenticatedAdmin, datamanager.deleteIndex)

Parametri

'/api/indexes/:index_name'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticatedAdmin
Funzione di *@passport* che verifica che l'utente sia autenticato e che il suo livello di accesso sia Amministratore.

datamanager.deleteIndex
Callback da richiamare al termine dell'autenticazione.

Descrizione
Intercetta la richiesta di cancellare un indice dal database di analisi. Verifica che essa sia autenticata come amministratore e richiama il metodo *deleteIndex* del *@datamanager*.

Gestione Login

post('/api/check/email', passport.checkNotAuthenticated, usermanager.checkMail)

Parametri

'/api/indexes/:index_name'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkNotAuthenticated Funzione di *@passport* che verifica che l'utente non sia autenticato.

usermanager.checkMail
Callback da richiamare al termine dell'autenticazione.

Descrizione Gestisce la richiesta di controllo della presenza di una particolare email già registrata al sistema. La condizione per richiedere questa informazione è che l'utente non sia autenticato, da qui la chiamata alla funzione *passport.checkNotAuthenticated*. Nel caso la chiamata alla precedente funzione abbia successo, viene richiamato il metodo *checkMail* di *@DatabaseAnalysisManager*.

post('/api/signup', passport.checkNotAuthenticated, usermanager.userSignup)

Parametri

'/api/signup'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkNotAuthenticated Funzione di *@passport* che verifica che l'utente non sia autenticato.

usermanager.userSignup
Callback da richiamare al termine dell'autenticazione.

Descrizione Gestisce la richiesta di registrazione da parte di un utente non autenticato. Se l'utente associato alla richiesta non è autenticato, viene richiamato il metodo *userSignup* di *@DatabaseAnalysisManager*.

get('/loggedin', function(req, res) { res.send(req.isAuthenticated() ? req.user : '0'); })

Parametri

'/loggedin'
Stringa contenente l'indirizzo della API da richiamare.

function(req, res) { res.send(req.isAuthenticated() ? req.user : '0'); }
Funzione anonima che richiama il metodo isAuthenticated sul richiedente. Se il metodo ritorna valore booleano **true**, viene ritornato al richiedente il proprio campo *user*, altrimenti ritorna 0.

Descrizione
Gestisce la richiesta di controllo dell'avvenuta autenticazione di un utente. Viene richiamato sul richiedente il metodo locale *isAuthenticated()*. Se l'utente è autenticato, gli vengono inviate le proprie informazioni associate, altrimenti viene ritornato 0.

post('/api/login', passport.checkNotAuthenticated, passport.authenticate, function(req, res) { res.send(req.user); })

Parametri

'/api/login'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkNotAuthenticated
Funzione di *@passport* che verifica che l'utente non sia autenticato.

passport.authenticate
Funzione di *@passport* che esegue l'autenticazione.

function(req, res) { res.send(req.user); }
Funzione anonima che manda al richiedente la risposta contenente il suo campo *user*.

Descrizione
Gestisce le richieste di autenticazione al Server. L'utente che richiede questa operazione non deve essere autenticato ma deve essere presente nel database degli utenti presente nel Server. Nel caso di risposta affermativa ad entrambe le precedenti condizioni, viene creata una nuova sessione per l'utente richiedente.

get('/api/logout', passport.checkAuthenticated, function(req, res) { req.logout(); res.send(200); })

Parametri

'/api/logout'
Stringa contenente l'indirizzo della API da richiamare.

passport.checkAuthenticated
Funzione di *@passport* che verifica che l'utente sia autenticato.
function(req, res) { req.logout(); res.send(200); }
Funzione anonima che effettua la disconnessione del richiedente *req* e manda al Client il codice *http* 200.

Descrizione
Gestisce le richieste di disconnessione dal sistema. Per effettuare questa operazione, l'utente deve essere autenticato. In caso affermativo, viene richiesto il *logout()* del richiedente.

Gestione richieste illecite

get('*', function(req, res) { res.sendfile(path.join(config.static_assets.dir, 'index.html')); })

Parametri
'*'
Stringa contentente l'indirizzo delle API da richiamare. In questo caso, l'idirizzo è una stringa qualsiasi.
function(req, res) { res.sendfile(path.join(config.static_assets.dir, 'index.html')); }
Funzione anonima che reindirizza il richiedente alla pagina iniziale della view del Client.

Descrizione La presente funzione gestisce ogni richiesta di pagina o file che il Server non possiede. L'argomento * della funzione *get* consente di gestire tutte le richieste che non sono state servite dalle precedenti funzioni del *@dispatcher*. Questa funzione viene richiamata soltanto quando non le funzioni precedenti non sono state chiamate.

Inizializzazione

dispatcherInit(app)

Parametro
app
Applicazione di *Express*.

Descrizione
Si occupa dell'inizializzazione del *@dispatcher*. Viene esportato come init().

4.1.2 @frontController

Descrizione
Il frontController è l'interfaccia di connessione tra il Server ed il Client. Si occupa dell'inizializzazione del *@dispatcher* reindirizzando ad esso le richieste del Client.

Dipendenze

- @dispatcher.

Metodi

initFrontController(app)

Parametro

app
Oggetto contenente l'applicazione di *Express*.

Descrizione
È la funzione di inizializzazione del *@frontController* ed indica di reindirizzare le richieste del Client al *@dispatcher* per gestirle. Viene esportato come init().

4.1.3 @index

Descrizione
Controller che si occupa dell'inizializzazione del *@frontController*.

Dipendendenze

- @frontController.

Metodi

init(app)

Parametro

app
Oggetto contenente l'applicazione di *Express*.

Descrizione
È la funzione di inizializzazione dell'*@index*, si occupa di inizializzare il *@frontController*.

4.1.4 @passport

Descrizione
Controller che si occupa dell'inizializzazione e gestione del modulo passort per le autenticazioni.

Dipendendenze

- passport;
- passport-local.Strategy;
- @MongooseDBFramework.

Metodi

authenticate()

Descrizione
Imposta il metodo di autenticazione, nel nostro caso come local.

initPassport(app)

Parametro

app
Oggetto contenente l'applicazione di *Express*.

Descrizione
Inizializza il controller *@passport*, inizializzando e avviando la sessione del modulo *@passport* e definisce le strategie di autenticazione.

checkAuthenticatedAdmin(req, res, next)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

res
Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

next
Funzione da invocare nel caso di successo.

Descrizione
Verifica che la richiesta proveniente dal client sia autenticata e che il livello dell'utente sia 1.

checkAuthenticated(req, res, next)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

res
Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente. La risposta al Client.

next
Funzione da invocare nel caso di successo.

Descrizione
Verifica che la richiesta proveniente dal client sia autenticata.

checkNotAuthenticated(req, res, next)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

res
Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

next
Funzione da invocare nel caso di successo.

Descrizione
Verifica che la richiesta proveniente dal client non sia autenticataa.

4.2 ::ModelServer

Package che racchiude i database, le loro operazioni di accesso e i file DSL con i relativi interpreti.

4.2.1 ::Database

4.2.1.1 @index

Descrizione
Si occupa dell'inizializzazione dei database degli utenti e di analisi. Crea l'amministratore di default di *MaaP*.

Dipendendenze

- mongoose;

- @MongooseDBAnalysis;
- @MongooseDBFramework;
- @DataRetrieverUsers.

Metodi

initDB(app)

Parametro

app
Oggetto contenente l'applicazione di *Express*.

Descrizione
Si occupa dell'inizializzazione dei database. Effettua la connessione al database utenti e a quello di analisi. Inizializza lo userDBManager, invoca addAdminDefault e inizializza il *@dataDBManager*. Viene esportata come init().

addAdminDefault(config, userDB))

Parametri

config
Oggetto contenente l'applicazione di Express.

userDB
Connessione a database degli utenti.

Descrizione
Si occupa dell'aggiunta dell'amministratore di default nel database degli utenti.

4.2.1.2 @MongooseDBAnalysis

Descrizione
Si occupa dell'inizializzazione dei modelli di *Mongoose*.

Dipendendenze

- fs;
- path;
- mongoose.

Metodi

init(app)

Parametro

app
Oggetto contenente l'applicazione di *Express*.

Descrizione
Legge la cartella contenente le *collectionData* dei DSL e per ogni file inizializza il modello di *Mongoose* come elemento dell'Array *modelArray* che poi esporta come model.

4.2.1.3 @MongooseDBFramework

Descrizione
Si occupa dell'inizializzazione dei modelli di *Mongoose*.

Dipendendenze

- mongoose.

Metodi

init(app)

Parametro

app
Oggetto contenente l'applicazione di *Express*.

Descrizione
Si occupa di creare gli schemi di *Mongoose* per la Collection utenti e la Collection query, esporta tali modelli come users e query e la connessione al database come connection.

4.2.2 ::DataManager

Package che gestisce gli accessi ai database.

4.2.2.1 ::DatabaseAnalysisManager

Package che gestisce le query ai database di analisi.

4.2.2.1.1 @DatabaseAnalysisManager

Descrizione
Modulo del ModelServer che si occupa di interpretare i dati richiesti e spediti da e al database di analisi. Prepara le query da effettuare e contatta il *@DataRetrieverAnalysis* per ottenere i dati voluti.

Dipendendenze

- path;
- @DataRetrieverAnalysis;
- @IndexManager;
- @JSonComposer.

Metodi

sendCollectionList(req, res)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

res
Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

Descrizione

Richiede al *@DataRetrieverAnalysis* l'elenco delle Collection e richiama il metodo di creazione dell'oggetto Collection del *@JSonComposer*. Infine ritorna l'oggetto CollectionList ottenuto.

sendCollection(req, res)

Parametri

req

Oggetto contenente i parametri e le informazioni del Client richiedente.

res

Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

Descrizione

Esegue una chiamata asincrona al *@DataRetrieverAnalysis* per ottenere una particolare Collection. Nel caso la Collection esista, viene richiamato il metodo *createCollection* di *@JSonComposer* per inviare al richiedente l'oggetto Collection desiderato. In caso la Collection non esista nel Server, viene ritornato il codice di errore *http* 404 al Client.

sendDocument(req, res)

Parametri

req

Oggetto contenente i parametri e le informazioni del Client richiedente.

res

Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

Descrizione

Esegue una chiamata asincrona al *@DataRetrieverAnalysis* per ottenere un particolare Document. Nel caso il Document esista, viene richiamato il metodo *createDocument* di *@JSonComposer* per inviare al richiedente l'oggetto Document desiderato. In caso il Document non esista nel Server, viene ritornato il codice di errore *http* 404 al Client.

sendDocumentEdit(req, res)

Parametri

req

Oggetto contenente i parametri e le informazioni del Client richiedente.

res

Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

Descrizione

Esegue una chiamata asincrona al *@DataRetrieverAnalysis* per ottenere un particolare Document modificabile. Nel caso il Document esista, viene richiamato il metodo *createDocument* di *@JSonComposer* per inviare al richiedente l'oggetto Document desiderato. In caso il Document non esista nel Server, viene ritornato il codice di errore *http* 404 al Client.

updateDocument(req, res)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

res
Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

Descrizione
Esegue una chiamata asincrona al metodo *updateDocument* di *@DataRetrieverAnalysis* per modificare un Document esistente nel Server. Nel caso la modifica effettuata dal *@DataRetrieverAnalysis* vada a buon fine, viene ritornato al Client un codice *http* di successo 200, altrimenti viene mandato il codice di errore *http* 404.

removeDocument(req, res)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

res
Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

Descrizione
Esegue una chiamata asincrona al metodo *removeDocument* di *@DataRetrieverAnalysis* per modificare un Document esistente nel Server. Nel caso la rimozione effettuata dal *@DataRetrieverAnalysis* vada a buon fine, viene ritornato al Client un codice *http* di successo 200, altrimenti viene mandato il codice di errore *http* 404.

4.2.2.1.2 @DataRetrieverAnalysis

Descrizione
Modulo che dialoga con il database di analisi per mezzo dell'infrastruttura *Mongoose*. Utilizza i file derivati dall'interpretazione del linguaggio DSL per richiedere e inviare i dati che l'utente sviluppatore decide di visualizzare e modificare.

Dipendenze

- @IndexManager.

Metodi

getModel(collection_name)

Parametro

collection_name
Stringa contenente il nome della Collection da ricercare nel database di analisi.

Dipendenza

- @MongooseDBAnalysis.

Descrizione
Ricerca nei modelli del database di analisi la Collection di nome *collection_name*. Se essa è presente, ne ritorna il modello, altrimenti ritorna l'intero -1.

getDocuments(model, querySettings, populate, callback)

Parametri

model
Modello della Collection di cui importare i Document.

querySettings
Oggetto contenente i parametri della query di ricerca dei Document.

populate
Oggetto contenente le funzioni populate specificate nel DSL della Collection.

callback
Funzione da richiamare al termine della funzione *getDocuments*.

Descrizione
Viene effettuata una query sul modello *model* passato come argomento alla funzione. Si utilizzano come opzioni di ricerca i campi dell'oggetto *querySettings*. Viene in questo modo creato un oggetto locale che verrà passato all fine dell'esecuzione della funzione alla *callback* come argomento. Se non pervengono risultati dalla query, l'oggetto passato sarà vuoto. Nel caso il risultato della query non sia nullo, viene poi effettuata l'operazione di popolamento specificata dal parametro *populate*. Infine viene richiamata la *callback* passando come argomento l'oggetto locale contenente il risulatato della query e del populate.

getCollectionsList()

Dipendenza
- DSL/collectionData/collectionsList.json.

Descrizione
Ritorna la lista delle Collection visibili dopo l'interpretazione del DSL.

applyTransformations(type, documentsArray, dslArray)

Parametri

type
Stringa contenente il tipo di visualizzazione dei Document presenti in *documentsArray*.

documentsArray
Array di Document a cui applicare le trasformazioni.

dslArray
Array di DSL legati alla Collection contenente i Document di *documentsArray*.

Dipendenza
- file dinamico contenente le trasformazioni di una Collection.

Descrizione
Scorre l'Array di DSL fino a trovare una trasformazione da applicare a uno o più dati dei Document. Se esiste almeno una trasformazione da applicare, la applica ad ogni Document contenuto nel *documentsArray*.

sortDocumentsByLabels(documents, keys)

Parametri

documents
Array di Document.

keys
Array di chiavi di un Document.

Descrizione

Ordina le coppie chiave-valore dei Document presenti nell'Array *documents* secondo l'ordine stabilito nell'array di chiavi *keys*.

getCollectionIndex(collection_name, column, order, page, callback)

Parametri

collection_name
Stringa contenente il nome della Collection.

column
Stringa contenente il nome della colonna da utilizzare per ordinare i Document.

order
Stringa contenente l'ordinamento da applicare ai Document, ascendente o discendente.

page
Numero intero che definisce il numero della pagina di Document da visualizzare.

callback
Funzione da richiamare al termine della funzione *getCollectionIndex*.

Dipendenza

- Oggetto dinamico derivante dall'analisi del file DSL associato alla Collection di nome *collection_name*.

Descrizione

Viene richiesto il modello della Collection di nome *collection_name*. Successivamente vengono costruiti gli Array delle etichette e dei dati della Collection. Il nome delle varie etichette viene stabilito dal file DSL associato alla Collection, altrimenti viene preso il nome di default della colonna. A questo punto viene eseguito un controllo per stabilire se il campo _id dei vari Document della Collection deve essere visibile nella visualizzazione. Viene eseguita una query nel database di analisi per ottenere tutti i Document che rispettano i parametri. Viene poi eseguita un'ulteriore query per creare localmente l'oggetto Collection con il numero di pagine specificato nel file DSL della Collection e per ottenere i Document identificati dalla pagina di visualizzazione richiesta dal Client. Se le query hanno portato ad un risultato positivo, viene richiamata la funzione *callback* con argomento il JSON creato localmente. Altrimenti la *callback* viene richiamata con argomento vuoto.

getDocumentShow(collection_name, document_id, callback)

Parametri

collection_name
Stringa contenente il nome della Collection di cui fa parte il Document identificato con *document_id*.

document_id
Id che identifica un singolo Document all'interno della Collection di nome *collection_name*.

callback
Funzione da richiamare al termine della funzione *getDocumentShow*.

Dipendenza

- JSON generato dall'inerpretazione del DSL della Collection di nome *collection_name*.

Descrizione

Richiede il modello della Collection *collection_name* e il file DSL associato. Viene controllato se l'output del DSL applica delle modifiche alla visualizzazione standard. Se ciò avviene, vengono impostate le etichette del JSON da inviare al Client come specificato nel file DSL. Inoltre vengono interpretate eventuali istruzioni di *populate* da applicare prima di effettuare la query. Infine, nel caso la query abbia successo, viene richiamata la funzione *callback* con argomento il JSON ottenuto dalla query. Se invece la query non è andata a buon fine, viene chiamata la *callback* con argomento vuoto.

getDocumentShowEdit(collection_name, document_id, callback)

Parametri

collection_name
Stringa contenente il nome della Collection di cui fa parte il Document identificato con *document_id*.

document_id
Id che identifica un singolo Document all'interno della Collection di nome *collection_name*.

callback
Funzione da richiamare al termine della funzione *getDocumentShowEdit*.

Dipendenza

- JSON generato dall'inerpretazione del DSL della Collection di nome *collection_name*.

Descrizione

Richiede il modello della Collection *collection_name* e il file DSL associato. Viene controllato se l'output del DSL applica delle modifiche alla visualizzazione standard. Se ciò avviene, vengono impostate le etichette del JSON da inviare al Client come specificato nel file DSL. I campi compositi del database di analisi vengono processati e viene inviato solamente il campo *id* dei campi compositi. Inoltre vengono interpretate eventuali istruzioni di *populate* da applicare prima di effettuare la query. Infine, nel caso la query abbia successo, viene richiamata la funzione *callback* con argomento il JSON ottenuto dalla query. Se invece la query non è andata a buon fine, viene chiamata la *callback* con argomento vuoto.

updateDocument(collection_name, document_id, newDocumentData, callback)

Parametri

collection_name
Stringa contenente il nome della Collection contenente il Document da modificare.

document_id
Id del Document da modificare.

newDocumentData
JSON contenente i campi da modificare nel Document desiderato.

callback
Funzione da richiamare al termine della modifica del Document.

Descrizione
Richiede il modello della Collection nominata *collection_name*. Successivamente richiama il metodo *update* sul modello della Collection scelta per aggiornare il Document identificato con *document_id* con le nuove informazioni di *newDocumentData*. Al termine di questa operazione, viene chiamata la *callback*.

removeDocument(collection_name, document_id, callback)

Parametri

collection_name
Stringa contenente il nome della Collection contenente il Document da eliminare.

document_id
Id del Document da eliminare.

callback
Funzione da richiamare al termine dell'eliminazione del Document.

Descrizione
Richiede il modello della Collection nominata *collection_name*. Successivamente richiama il metodo *remove* sul modello della Collection scelta per rimuovere il Document identificato con *document_id*. Al termine di questa operazione, viene chiamata la *callback*.

4.2.2.2 ::DatabaseUserManager

Package che gestisce gli accessi al database utenti.

4.2.2.2.1 @DatabaseUserManager

Descrizione
Modulo del ModelServer che si occupa di interpretare i dati richiesti e spediti da e al database utenti. Prepara le query da effettuare e contatta il *@DataRetrieverUsers* per ottenere i dati voluti. Inoltre dialoga direttamente con *@MongooseDBFramework* per l'esecuzione di test durante la registrazione.

Dipendendenze

- path;
- @DataRetrieverUsers;
- @MongooseDBFramework;
- @JSonComposer.

Metodi

checkMail(req, res)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

res
Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

Descrizione
Effettua un controllo di presenza nel database utenti della email specificata dal Client. In caso di presenza, viene inviato al client un errore *http* 400. In caso contrario, viene segnalato all'utente la mancanza di un indirizzo email già registrato e viene inviato il codice *http* 304.

userSignup(req, res)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

res
Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

Descrizione
Richiama il metodo *addUser* di *@DataRetrieverUsers* per registrare un nuovo utente al sistema. Vengono passati al metodo *addUser* le credenziali contenute nell'oggetto *req*. In caso di avvenuto inserimento, viene inviato al Client il codice di conferma *http* 200. In caso contrario, viene inviato il codice di errore *http* 404.

sendUserProfile(req, res)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

res
Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

Descrizione
Richiama il metodo *getUserProfile* di *@DataRetrieverUsers* per ottenere l'oggetto utente sulla base dei parametri inviati da *req*. Successivamente invia a *res* il JSON dell'utente creato tramite il metodo *createUserProfile* di *JsonComposer*.

sendUserProfileEdit(req, res)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

res
Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

Descrizione Richiama il metodo *getUserProfile* di *@DataRetrieverUsers* per ottenere l'oggetto utente sulla base dei parametri inviati da *req*. Successivamente invia a *res* il JSON dell'utente creato tramite il metodo *createUserProfileEdit* di *@JsonComposer*.

updateUserProfile(req, res)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

res
Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

Descrizione
Richiama il metodo *updateUserProfile* di *@DataRetrieverUsers* passando come parametro il richiedente *req* del Client. Ritorna al Client un codice *http* 200 in caso di riuscita, un codice 400 altrimenti.

getUsersList(req, res)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

res
Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

Descrizione
Richiama il metodo *getUsersList* di *@DataRetrieverUsers* utilizzando i parametri di visualizzazione di *req* o assegnando dei valori di default. Ottenuti i dati, risponde a *res* inviando il risultato della chiamata del metodo *createUsersList* di *@JsonComposer*.

sendUser(req, res)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

res
Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

Descrizione
Richiama il metodo *getUserProfile* di *@DataRetrieverUsers* passando come parametro il campo *user_id* di *req*. Risponde a *res* inviando il risultato della chiamata del metodo *createUser* di *@JsonComposer*.

sendUserEdit(req, res)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

res
Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

Descrizione
Richiama il metodo *getUserProfile* di *@DataRetrieverUsers* passando come parametro il campo *user_id* di *req*. Risponde a *res* inviando il risultato della chiamata del metodo *createUser* di *@JsonComposer*.

updateUser(req, res)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

res
Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

Descrizione
Richiama il metodo *updateUser* di *@DataRetrieverUsers* passando come parametro il richiedente *req*. Ritorna al Client un codice *http* 200 in caso di riuscita, un codice 400 altrimenti.

removeUser(req, res)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

res
Oggetto duale a *req* su cui invocare i metodi per inviare informazioni al Client richiedente.

Descrizione
Richiama il metodo *removeUser* di *@DataRetrieverUsers* passando come parametro il campo *email* del richiedente *req*. Ritorna al Client un codice *http* 200 in caso di riuscita, un codice 400 altrimenti.

4.2.2.2.2 @DataRetrieverUsers

Descrizione
Modulo che dialoga direttamente con *@MongooseDBFramework* per effettuare le query di visualizzazione e modifica dei dati contenuti nel database degli utenti.

Dipendenze

- @MongooseDBFramework.

Metodi

addUser(email, password, level, callback)

Parametri

email
Stringa contenente l'indirizzo email del nuovo utente.

password
Stringa contenente la password del nuovo utente.

level
Numero intero che identifica il livello di permesso di un utente. Il numero 0 identifica un utente standard, 1 individua un utente admin.

callback
Funzione da richiamare alla fine dell'inserimento.

Descrizione
La funzione inserisce un nuovo utente nel database utenti del sistema. Alla fine dell'inserimento, se non sono sollevati errori richiama la funzione *callback* assegnando come parametro **true**, altrimenti assegna il parametro **false**.

getUserProfile(user_id, callback)

Parametri

user_id
Id univoco di un utente registrato al sistema.

callback
Funzione da richiamare alla fine dell'ottenimento del profilo utente.

Descrizione
Viene ricercato nel database utenti un utente registrato con id uguale a *user_id*. Nel caso venga trovato, viene richiamata la funzione *callback* passando come parametro l'utente trovato, altrimenti viene chiamata con un oggetto vuoto.

updateUserProfile(req, callback)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

callback
Funzione da richiamare alla fine dell'aggiornamento del profilo utente.

Descrizione
Vengono estratti i dati da aggiornare dall'oggetto *req* del richiedente. Successivamente viene effettuata una query di aggiornamento sul database degli utenti con i dati forniti da *req*. In caso l'aggiornamento dei dati avvenga con successo, viene chiamata la funzione *callback* con argomento **true**, altrimenti viene chiamata con argomento **false**.

getUsersList(column, order, page, perpage, callback)

Parametri

column
Stringa contenente il nome della colonna su cui ordinare gli utenti.

order
Stringa che specifica l'ordinamento ascendente o discendente degli utenti.

page
Numero intero che specifica il numero della pagina di utenti da visualizzare.

perpage
Numero intero che specifica la quantità di utenti da visualizzare per pagina.

callback
Funzione da richiamare alla fine dell'ottenimento dei profili utente.

Descrizione
Viene innanzitutto importata l'intera lista degli utenti del sistema. Successivamente, se il risultato della query è vuoto, viene richiamata la funzione *callback* come argomento un oggetto vuoto. Altrimenti, viene processata l'intera lista al fine di invocare la *callback* specificando come argomento l'insieme di Document desiderato. I Document saranno quelli che corrispondono alla pagina *page* correntemente richiesta con l'ordinamento *order* desiderato sulla colonna scelta *column*.

updateUser(req, callback)

Parametri

req
Oggetto contenente i parametri e le informazioni del Client richiedente.

callback
Funzione da richiamare alla fine dell'aggiornamento del profilo utente.

Descrizione
Vengono estratti i dati da aggiornare dal richiedente *req* inerenti all'utente da aggiornare. Successivamente viene effettuata la query di aggiornamento sul modello del database utenti estratto. Infine viene richiamata la funzione *callback* con argomento **true** se l'aggiornamento è andato a buon fine, l'argomento è **false** altrimenti.

removeUser(email, callback)

Parametri

email
Stringa contenente l'indirizzo email dell'utente che si desidera eliminare dal database.

callback
Funzione da richiamare alla fine dell'eliminazione del profilo utente.

Descrizione
Viene estratto il modello degli utenti su cui verrà invocato il metodo *remove* per eliminare l'utente con campo email uguale al parametro *email* passato alla funzione. Al termine della query di eliminazione, viene richiamata la funzione *callback* con argomento **true** se l'aggiornamento è andato a buon fine, l'argomento è **false** altrimenti.

4.2.2.3 ::IndexManager

Package realizzato per creare e gestire gli indici utilizzati nel database di analisi.

4.2.2.3.1 @IndexManager

Descrizione
Modulo del ModelServer che si occupa di creare e gestire gli indici utilizzati nel database di analisi.

Metodi

getModel(collection_name)

Parametro

collection_name
Stringa contenente il nome della Collection di cui ottenere il modello.

Dipendenza

- @MongooseDBAnalysis.

Descrizione
Richiede al modulo *@MongooseDBAnalysis* il modello della Collection di nome *collection_name*. Se la Collection è presente, la funzione *getModel* ritorna il suo modello, altrimenti ritorna -1.

addQuery(collection_name, select)

Parametri

collection_name
Stringa contenente il nome della Collection su cui è stata effettuata la query.

select
Array contenente i campi visualizzati della Collection.

Dipendenza

- @MongooseDBFramework.query.

Descrizione
Viene richiesto l'insieme delle query effettuate sul database di analisi precedentemente alla query corrente. Successivamente, se la query è stata eseguita per la prima volta, viene inserita nell'insieme delle query effettuate. Se invece la query è stata già eseguita, il suo contatore viene incrementato.

resetQueries(callback)

Parametro

callback
Funzione da richiamare alla fine della funzione *resetQuery*.

Dipendenze

- @MongooseDBFramework.query;
- @MongooseDBFramework.connection.

Descrizione
Richiede il modello della Collection delle query, su cui poi viene applicato il metodo *dropCollection* che resetta la Collection delle query nel database.

getQueries(page, perpage, n_elements, callback)

Parametri

page
Numero intero che indica la pagina delle query da visualizzare.

perpage
Numero intero che indica il numero di query da visualizzare per pagina.

n_elements
Numero intero che indica il numero di query totali da visualizzare.

callback
Funzione da richiamare alla fine della funzione *getQueries*.

Dipendenze

- @MongooseDBFramework;
- @MongooseDBFramework.query.

Descrizione
Richiede l'intera Collection delle query al database di analisi. Calcola il sottoinsieme di query da visualizzare tenendo conto dell'indicatore della pagina da visualizzare e dagli elementi da visualizzare per pagina. Esegue quindi una nuova query per ottenere l'insieme delle query che rispettano suddetti parametri. Se avviene un errore o se non esistono query che rispettano i parametri scelti, viene passato alla *callback* un oggetto vuoto. Se invece la ricerca ha avuto esito positivo, viene richiamata la *callback* passando come parametro l'insieme delle query che soddisfano i requisiti.

createIndex(query_id, name_index, callback)

Parametri

query_id
Id identificativo della query da associare all'indice da creare.

name_index
Stringa contenente il nome dell'indice da creare.

callback
Funzione da richiamare alla fine della creazione dell'indice.

Dipendenze

- @MongooseDBFramework;
- @MongooseDBFramework.query.
- schema della Collection su cui la query di cui si vuole creare l'indice esegue le richieste.

Descrizione
Viene richiesto il modello della query, grazie a cui viene cercata nel database di analisi la query con campo _id uguale al parametro *_id*. Se la query non esiste o si riscontrano problemi di accesso al database, viene chiamata la funzione *callback* con argomento **false**. Altrimenti viene caricato lo schema della Collection su cui la query viene eseguita. Sulla collection viene creato l'indice voluto. Se l'operazione di creazione non ha successo, viene viene chiamata la funzione *callback* con argomento **false**, altrimenti viene chiamata la *callback* con argomento **true**.

4.2.2.4 @JsonComposer

Descrizione
Modulo che si occupa della preparazione e della formattazione degli oggetti JSON inviati dal Server al Client.

Metodi

createCollectionsList(collectionsList)

Parametro

collectionsList
Array di Collection.

Descrizione
La funzione riceve come parametro un Array di Collection, crea e ritorna l'oggetto JSON rappresentante l'insieme delle Collection.

checkLabels(labelsArray)

Parametro

labelsArray
Array di etichette di una Collection.

Descrizione Scorre l'Array di label e ritorna *true* se fra essi è presente il campo *_id* o il flag *__IDLABEL2SHOW__*, ritorna *false* altrimenti.

createCollection(labels, data, config)

Parametri

labels
Array contenente l'elenco delle etichette della Collection da creare.

data
Array contenente i dati sensibili della Collection.

config
Oggetto contenente impostazioni di configurazione.

Descrizione
Scorre l'intero Array di dati della Collection e salva i suoi elementi in un Array locale. Esegue un controllo sull'Array di label per identificare la presenza dell'etichetta *__IDLABEL2SHOW__*, la quale definisce di far visualizzare il campo *_id* nel Client. Dopo aver formattato correttamente gli elementi della Collection, ritorna il JSON corrispondente, aggiungendo l'oggetto *config* contenente le configurazioni dell'oggetto utili al Client.

createDocument(labels, data)

Parametri

labels
Array contenente l'elenco delle etichette del Document da creare.

data
Array contenente i dati sensibili del Document da creare.

Descrizione
Scorre l'intero Array di dati del Document e salva i suoi elementi in un Array locale. Esegue un controllo sull'Array di label per identificare la presenza dell'etichetta *__IDLABEL2SHOW__*, la quale definisce di far visualizzare il campo *_id* nel Client. Dopo aver formattato correttamente gli elementi del Document, ritorna il JSON corrispondente.

createQueriesList(queries, config)

Parametri

queries
Array contenente l'insieme delle query da creare.

config
Oggetto contenente impostazioni di configurazione.

Descrizione
Viene creato localmente l'Array contenente le etichette dell'oggetto lista di query. Viene successivamente scorso l'Array delle query e vengono copiati localmente i loro campi sensibili, per poi essere formattati correttamente. Infine viene realizzato e ritornato il JSON corrispondente, aggiungendo l'oggetto contenente le impostazioni di configurazione della lista di query.

createIndexesList(indexes)

Parametro

indexes
Array contenente l'insieme di indici da creare.

Descrizione
Viene creato localmente l'Array contenente le etichette dell'oggetto lista di indici. Viene successivamente scorso l'Array degli indici e vengono copiati localmente i loro campi sensibili, per poi essere formattati correttamente. Infine viene realizzato e ritornato il JSON corrispondente.

createUserProfile(user)

Parametro

user
Oggetto contenente i campi email e livello di accesso di un utente.

Descrizione
Viene interpretato il campo numerico *level* e convertito nella stringa corrispondente al livello di accesso dell'utente da creare. Viene infine ritornato il JSON corrispondente.

createUserProfileEdit(user)

Parametro

user
Oggetto contenente il campo email di un utente.

Descrizione
Viene ritornato il JSON di un utente modificabile formattato correttamente.

createUsersList(users, config)

Parametri users
Array di oggetti utente.

config
Oggetto contenente impostazioni di configurazione.

Descrizione
Viene creato localmente l'Array delle etichette della lista utenti. Successivamente viene copiato in un Array locale ogni utente e il campo *level* di ognuno viene interpretato e convertito nella stringa corrispondente. Infine viene assemblato il JSON della lista utenti e viene aggiunto l'oggetto contenente le impostazioni di configurazione della lista.

createUser(user)

Parametro

user
Oggetto contenente i campi id, email e livello di accesso di un utente.

Descrizione
Viene creato localmente lo scheletro dell'oggetto utente. Viene poi interpretato l'intero contenuto nel campo *level* dell'oggetto *user*. Viene poi ritornato il JSON dell'utente formattato correttamente.

4.2.3 ::DSL

4.2.3.0.1 @DSLManager

Descrizione

Modulo del *ModelServer* che si occupa controllare la presenza dei file DSL nell'apposita cartella definita nel file di configurazione e invoca il parser dei DSL.

Dipendenze

- fs;
- path;
- @DSLParser;
- @schemaGenerator.

Metodi

generateFunction(transformation)

Parametro

transformation

Oggetto che contiene il nome della trasformazione e il codice della trasformazione da eseguire.

Descrizione Si occupa di generare le funzioni JavaScript per gestire le trasformazioni dei dati definite nel DSL.

deleteFolderRecursive(path)

Parametro

path

Percorso della cartella da svuotare.

Descrizione

Si occupa di cancellare i file e le cartelle all'interno della cartella *path* specificata.

checkDSL(app)

Parametro

app

Oggetto contenente l'applicazione di *Express*.

Descrizione

Si occupa di verificare la correttezza del DSL scritto, verificando che tale file contenga un codice corretto sintatticamente e attraverso l'uso del *@DSLparser* esegui il parsing vero e proprio. Controlla che il risulatato del parsing sia in formato JSON, lo salva su file e utilizza il metodo *generateFunction* per creare le funzioni di trasformazione. Se non esiste, si occupa di generare lo schema *Mongoose* della Collection specificata nel file DSL.

4.2.3.0.2 @DSLParser

Descrizione

Modulo del ModelServer che si occupa effettuare il parsing di un DSL.

Dipendenze

- @JavascriptParser.

Metodi

addField(collection, field)

Parametri

collection
Array della struttura del JSON.

field
Campo da aggiungere alla struttura del JSON.

Descrizione
Inizializza il campo field nell'Array della struttura del JSON, dove tale campo è definito nella struttura di default del JSON, con il valore di defalt.

checkFieldThrow(collection, field, root)

Parametri

collection
Array della struttura del JSON.

field
Campo da verificare nella struttura del JSON.

root
È il nome della sezione in cui deve essere definito il campo.

Descrizione
Chiama il metodo *checkField* e se tale metodo ritorna **false** allora lancia un eccezione. Questo metodo serve per verificare che il campo sia definito.

checkField(collection, field)

Parametri

collection
Array della struttura del JSON.

field
Campo da verificare nella struttura del JSON.

Descrizione
Verifica se un campo è definito, se lo è ritorna **true**, altrimenti false.

checkFieldContentThrow(collection, field, root)

Parametri

collection
Array della struttura del JSON.

field
Campo da verificare nella struttura del JSON.

root
È il nome della sezione in cui deve essere definito il campo.

Descrizione
Serve per verificare che il campo sia definito e non vuoto. Utilizza il metodo *checkFieldContent* e se tale metodo ritorna false lancia un eccezione.

checkFieldContent(collection, field)

Parametri

collection
Array della struttura del JSON.

field
Campo da verificare nella struttura del JSON.

Descrizione
Verifica se un campo è definito e non vuoto, se lo è ritorna true, altrimenti false.

IntValue(value, field)

Parametri

value
Valore del campo.

field
Campo da verificare nella struttura del JSON.

Descrizione
Verifica che il valore sia numerico, se non lo è lancia un eccezione.

parseDSL(DSLstring)

Parametro

DSLstring
Contenuto del file maap di configurazione.

Descrizione
Si occupa di effettuare il parsing del DSL trasformandolo in JSON. Per effettuare il parsing verifica che i campi obbligatori siano definiti e non vuoti, che i campi il cui contenuto deve essere un numero lo sia e che la sintassi delle funzioni di trasformazione dei campi sia valida utilizzando il parser *@JavascriptParser*.

4.2.3.0.3 @index

Descrizione Modulo che si occupa dell'inizializzazione del *@DSLManager*.

Dipendendenze

- @DSLManager.

Metodi

init(app)

Parametro

app
Oggetto contenente l'applicazione di *Express*.

Descrizione
Si occupa di invocare il metodo *checkDSL* di *@DSLManager*.

4.2.3.0.4 @JavascriptParser

Descrizione
Modulo che effettua il parsing del codice JavaScript. Tale modulo non è stato scritto da noi ma generato con PEG.js. Si consulti il manuale di PEG.js per ulteriori dettagli.

4.2.3.0.5 @schemaGenerator

Descrizione
Modulo che si occupa dell'inizializzazione del *@DSLManager*.

Dipendenza

- fs.

Metodi

getPopulatedCollection(populateArray, key)

Parametri

populateArray
Array di oggetti contenenti la Collection su cui effettuare il populate e la chiave di riferimento.

key
Chiave da ricercare.

Descrizione
Verifica che sia presente all'interno di un array di oggetti un elemento con chiave uguale al parametro *key*, se la trova ritorna la Collection corrispondente.

arrayAddElement(element, array)

Parametri

element
Oggetto da aggiungere all'array.

array
Array su cui aggiungere l'elemento.

Descrizione
Verifica che l'oggetto non sia presente all'interno dell'Array e se non lo trova lo aggiunge.

generate(config, dslJson)

Parametri

config
Contiene la configurazione dell'ambiente *MaaP*.

dslJson
Contiene il JSON del parsing del DSL.

Descrizione
Si occupa di generare lo schema della Collection per mongoose partendo dai tipi di dati specificati dall'utente nel DSL.

5 Specifica componenti Maap::Client

5.1 ::View

5.1.1 ::Template

I template utilizzati da AngularJs per la generazione delle pagine sono i seguenti:

Template	Pagina
collection.html	Pagina contenente i documenti di una determinata collection
dashboard.html	Pagina principale visualizzata dopo il login
document.html	Pagina per la visualizzazione di un singolo document
documentEdit.html	Pagina per la modifica di un singolo document
indexCollection	Pagina per la visualizzazione degli indici creati (admin)
login.html	Pagina per il login
navbar.html	Barra di navigazione superiore, inclusa nella maggior parte delle pagine
queryCollection.html	Pagina per la visualizzazione delle query piú utilizzate (admin)
register.html	Pagina per la registrazione
userCollection.html	Pagina per la visualizzazione della collection degli utenti maap (admin)
userEdit.html	Pagina per la modifica del profilo dell'utente (user) / di un utente (admin)
userProfile	Pagina per la visualizzazione del profilo utente (user) / di un utente (admin)

5.2 ::ControllerModelView

5.2.1 ::ControllerClient

5.2.1.1 @CollectionController

Descrizione
Modulo che descrive il Controller della Collection view.

Utilizzo
Controller utilizzato da AngularJS per fornire e gestire i dati della Collection view. Fornisce l'inizializzazione di base della pagina ed esegue la mediazione tra la Collection view e le richieste di visualizzazione e modifica dei dati di una Collection.

Dipendendenze iniettate al controller

- $scope;
- $route;
- $location;
- $routeParams;
- DocumentEditService;
- CollectionDataService;

Attributi

- $scope.current_sorted _column;
- $scope.column _original _name;
- $scope.current _sort;
- $scope.current _page;
- $scope.current_collection;
- $scope.rows;

Funzionalità

init()

Descrizione

Funzione di inizializzazione del CollectionController. Imposta i valori iniziali delle variabili dello $scope associate alla Collection view ed esegue la funzione *getData()*.

getData()

Descrizione

Richiama la richiesta RESTful *query* sulla $resource fornita dal CollectionDataService. Con questa funzione vengono prelevati i dati legati alla Collection da visualizzare e vengono passati alla Collection view tramite il binding con lo $scope.

$scope.numerify(num)

Parametro

num

Intero rappresentante il numero di pagine da visualizzare.

Descrizione

Funzione ausiliaria che crea un array di dimensione num che verrà poi usato dalla directive ng-repeat di angular per la creazione del numero di pagine disponibili.

$scope.previousPage()

Descrizione

Aggiunge allo $scope la funzionalità di passaggio dalla pagina di visualizzazione corrente alla precedente. Diminuisce di 1 l'indicatore della pagina corrente e esegue una nuova richiesta al server per i dati corrispondenti alla pagina precedente con *getData()*.

$scope.nextPage()

Descrizione

Aggiunge allo $scope la funzionalità di passaggio dalla pagina di visualizzazione corrente alla successiva. Aumenta di 1 l'indicatore della pagina corrente e esegue una nuova richiesta al server per i dati corrispondenti alla pagina successiva con *getData()*.

$scope.toPage(index)

Parametro

index

Numero intero che definisce il numero della pagina da visualizzare.

Descrizione

Aggiunge allo $scope la funzionalità di passaggio alla pagina di visualizzazione specificata dal parametro *index*. Aggiorna l'indicatore della pagina corrente settandolo all'indice passato per parametro alla funzione. Successivamente esegue una richiesta al server per ottenere i dati per la nuova visualizzazione.

changeSort()

Descrizione

Cambia la variabile $current_sort dello $scope che regola l'ordine di visualizzazione dei Document, da discendente ad ascendente e viceversa.

$scope.columnSort(index)

Parametro

index

Numero intero che definisce il numero della colonna a cui applicare l'ordinamento.

Descrizione

Funzione che cambia l'ordinamento di visualizzazione di una colonna della Collection view.

Se la colonna in questione è una colonna diversa da quella che precedentemente ordinata, la variabile *current_sorted_column* dello $scope viene aggiornata, altrimenti la funzione cambia solamente l'ordinamento della colonna.

$scope.deleteDocument(index)

Parametro

index

Numero intero che definisce il numero del Document da eliminare.

Descizione

Funzione che elimina il Document identificato dall'indice *index*. Effettua una richiesta REST di rimozione alla risorsa pubblicizzata da DocumentEditService fornendo come argomenti l'identificativo della Collection correntemente visualizzata e l'indice del Document da eliminare.

Se l'eliminazione ha successo viene lanciato l'aggiornamento della vista corrente con il Document eliminato, altrimenti si richiama la callback di errore.

5.2.1.2 @DashboardController

Descrizione

Modulo che descrive il Controller della Dashboard.

Utilizzo

Controller utilizzato da AngularJS per fornire e gestire i dati della dashboard. Fornisce un immediato elenco delle collection disponibili.

Dipendendenze iniettate al controller

- $scope;
- CollectionListService;

Funzionalità

CollectionListService.get()

Descrizione

Funzione che richiede al server l'elenco delle collection disponibli e le inserisce nello scope.

5.2.1.3 @DocumentController

Descrizione

Modulo che descrive il Controller della pagina di visualizzazione di un documento.

Utilizzo

Controller utilizzato da AngularJS per fornire e gestire i dati durante la visualizzazione di un document.

Dipendendenze iniettate al controller

- $scope;
- $location
- DocumentDataService;
- DocumentEditService;
- $routeParams.

Attributi

- $scope.current_collection;
- $scope.current_document;
- $scope.values;

Funzionalità

DocumentDataService.query()

Parametri

$scope.current_collection;
$scope.current_document.
Parametri che identificano la collection e il document da visualizzare.

Descrizione

Funzione che richiede al server il document da visualizzare.

delete_document()

Parametri

$scope.current_collection;
$scope.current_document. Parametri che identificano la collection e il document da cancellare.

Descrizione

Funzione che richiede al server la cancellazione di un document.

5.2.1.4 @DocumentEditController

Descrizione
Modulo che descrive il Controller della pagina di modifica di un documento.

Utilizzo
Controller utilizzato da AngularJS per fornire e gestire i dati durante la modifica di un document.

Dipendendenze iniettate al controller

- $scope;
- $location
- DocumentEditService;
- $routeParams.

Attributi

- $scope.current_collection;
- $scope.current_document;
- $scope.original_data;
- $scope.original_keys;

Funzionalità

DocumentEditService.query()

Parametri
$scope.current_collection;
$scope.current_document.
Parametri che identificano la collection e il document da modificare.

Descrizione
Funzione che richiede al server il document da modificare.

edit_document()

Parametri
$scope.current_collection;
$scope.current_document.
Parametri che identificano la collection e il document da sovrascrivere.

Descrizione
Funzione che richiede al server di sovrascrivere il document identificato dai parametri con quello modificato dall'utente.

delete_document()

Parametri
$scope.current_collection;
$scope.current_document.
Parametri che identificano la collection e il document da cancellare.

Descrizione
Funzione che richiede al server la cancellazione di un document.

5.2.1.5 @IndexController

Descrizione
Modulo che descrive il Controller della pagina di gestione degli indici.

Utilizzo
Controller utilizzato da AngularJS per fornire e gestire i dati degli indici.

Dipendendenze iniettate al controller

- $scope;
- $route;
- $location;
- IndexService;

Attributi

- $scope.rows;

Funzionalità

init()

Descrizione
Funzione di inizializzazione dell'Index Controller. Imposta i valori iniziali delle variabili dello $scope ed esegue la funzione *getData()*.

getData()

Descrizione
Richiama la richiesta RESTful *query* sulla $resource fornita da IndexService. Con questa funzione vengono prelevati i dati legati agli indici da visualizzare e vengono passati alla view tramite il binding con lo $scope.

$scope.numerify(num)

Parametro

num
Numero intero che specifica il numero di pagine da visualizzare.

Descrizione
Funzione ausiliaria che crea un array di dimensione *num* che verrà poi usato dalla directive ng-repeat di angular per la creazione del numero di pagine disponibili.

$scope.previousPage()

Descrizione
Aggiunge allo $scope la funzionalità di passaggio alla pagina di visualizzazione precedente alla corrente. Diminuisce di 1 l'indicatore della pagina corrente e esegue una nuova richiesta al server per i dati corrispondenti alla pagina precedente con *getData()*.

$scope.nextPage()

Descrizione
Aggiunge allo $scope la funzionalità di passaggio alla pagina di visualizzazione successiva alla corrente. Aumenta di 1 l'indicatore della pagina corrente e esegue una nuova richiesta al server per i dati corrispondenti alla pagina successiva con *getData()*.

$scope.toPage(index)

Parametro

index

Numero intero che specifica il numero di pagine da visualizzare.

Descrizione

Aggiunge allo $scope la funzionalità di passaggio alla pagina di visualizzazione specificata dal parametro *index*. Aggiorna l'indicatore della pagina corrente settandolo all'indice passato per parametro alla funzione. Successivamente esegue una richiesta al server per ottenere i dati per la nuova visualizzazione.

changeSort()

Descrizione

Cambia la variabile $current_sort dello $scope che regola l'ordine di visualizzazione dei Document, da discendente ad ascendente e viceversa.

$scope.columnSort(index)

Parametro

index

Numero intero che specifica il numero di pagine da visualizzare.

Descrizione

Funzione che cambia l'ordinamento di visualizzazione di una colonna della Collection view.

Se la colonna in questione è una colonna diversa da quella che precedentemente ordinata, la variabile *current_sorted_column* dello $scope viene aggiornata, altrimenti la funzione cambia solamente l'ordinamento della colonna.

$scope.deleteDocument(index)

Parametro

index

Numero intero che specifica il numero di pagine da visualizzare.

Descrizione

Funzione che elimina l'indice identificato dall'indice *index*. Effettua una richiesta REST di rimozione alla risorsa pubblicizzata da IndexService fornendo come argomenti l'indice passato come argomento.

Se l'eliminazione ha successo viene lanciato l'aggiornamento della vista corrente con il Document eliminato, altrimenti si richiama la callback di errore.

5.2.1.6 @LoginController

Descrizione

Modulo che descrive il Controller della pagina di login.

Utilizzo

Controller utilizzato da AngularJS per fornire e gestire i dati durante l'autenticazione.

Dipendendenze iniettate al controller

- $scope;
- $route;
- $cookieStore;

- $location;
- AuthService;

Attributi

- $scope.credentials;

Funzionalità

$scope.login()

Parametro

$scope.credentials
Credenziali di accesso inserite dall'utente.

Descrizione
Funzione che trasmette al server le credenziali inserite dall'utente e verifica se sono corrette. In caso positivo autentica l'utente, altrimenti mostra un errore.

5.2.1.7 @NavBarController

Descrizione
Modulo che descrive il Controller della barra di navigazione superiore.

Utilizzo
Viene utilizzato per gestire i link e le informazioni contenute sulla barra di navigazione presente in tutte le pagine protette da autenticazione. Contiene i link alle varie collection e i pulsanti per la gestione del profilo utente e per il logout.

Dipendendenze iniettate al controller

- $scope;
- $route;
- $cookieStore;
- $location;
- LogoutService;
- CollectionListService.

Attributi

- $scope.isAdmin;
- $scope.singup_enabled.

Funzionalità

logout()

Descrizione
Richiama la richiesta RESTful *logout* sulla $resource fornita da LogoutService. Con questa funzione viene distrutta la sessione riguardante l'utente presente sul server e si viene poi disconnessi.

5.2.1.8 @ProfileController

Descrizione
Modulo che descrive il Controller della pagina di gestione del profilo.

Utilizzo
Controller utilizzato da AngularJS che permette all'utente di gestire il suo profilo, visualizzandone le informazioni contentute.

Dipendendenze iniettate al controller

- $scope;
- $location;
- ProfileDataService;
- ProfileEditService;

Attributi

- $scope.original_data;
- $scope.original_keys;

Funzionalità

ProfileDataService.query()

Descrizione
Funzione che richiede al server le informazioni riguardanti il profilo dell'utente autenticato.

$scope.deleteDocument()

Descrizione
Funzione che elimina dal server il profilo dell'utente autenticato, cancellando l'utente dal sistema.

5.2.1.9 @ProfileEditController

Descrizione
Modulo che descrive il Controller della pagina di modifica di un documento.

Utilizzo
Controller utilizzato da AngularJS che permette all'utente di gestire il suo profilo, visualizzando e modificando le informazioni contentute.

Dipendendenze iniettate al controller

- $scope;
- $location
- ProfileEditService;

Attributi

- $scope.oldPassword;

- $scope.newPassword1;
- $scope.newPassowrd2;
- $scope.original_data;
- $scope.original_keys;
- $scope.valid.

Funzionalità

ProfileEditService.query()

Descrizione
Richiama la richiesta RESTful *query* sulla $resource fornita da ProfileEditService. Con questa funzione vengono prelevati i dati legati al profilo utente e caricati sullo scope.

edit_document()

Parametro

json_data
JSON contenente le nuove informazioni da modificare nel database utenti presente nel Server.

Descrizione
Funzione che invia al server il profilo modificato nel client. Assembla un JSON a partire da original_keys e original_data. Quest'ultima variabile nel frattempo è stata modificata con i nuovi valori del documento.

$scope.deleteDocument()

Descrizione
Funzione che elimina dal server il profilo dell'utente autenticato, cancellando l'utente dal sistema.

5.2.1.10 @QueryController

Descrizione
Modulo che descrive il Controller della pagina di gestione delle query.

Utilizzo
Controller utilizzato da AngularJS per fornire e gestire i dati della pagina di gestione delle query.

Dipendendenze iniettate al controller

- $scope;
- $route;
- $location;
- QueryService;
- IndexService;

Attributi

- $scope.current_sorted_column;
- $scope.column_original_name;

- $scope.current_sort;
- $scope.current_page;
- $scope.current_collection;
- $scope.rows;

Funzionalità

init()

Descrizione
Funzione di inizializzazione del QueryController. Imposta i valori iniziali delle variabili ed esegue la funzione *getData()*.

getData()

Descrizione
Richiama la richiesta RESTful *query* sulla $resource fornita dal QueryService. Con questa funzione vengono prelevati i dati legati alla Collection da visualizzare e vengono passati alla Collection view tramite il binding con lo $scope.
Se non vengono visualizzati risultati, ovvero se la Collection non esiste, avviene un redirect alla pagina che mostra l'errore 404.

$scope.createIndex(id)

Parametro
id
Id della query su cui creare l'indice.

Descrizione
Funzione ausiliaria che crea un indice a partire da una delle query presenti nella collection delle query.

$scope.numerify(num)

Parametro
num
Intero rappresentante il numero di pagine disponibili.

Descrizione
Funzione ausiliaria che crea un Array di dimensione *num* che verrà poi usato dalla directive *ng-repeat* di angular per la creazione del numero di pagine disponibili.

$scope.previousPage()

Descrizione
Aggiunge allo $scope la funzionalità di passaggio alla pagina di visualizzazione precedente alla corrente. Diminuisce di 1 l'indicatore della pagina corrente e esegue una nuova richiesta al server per i dati corrispondenti alla pagina precedente con *getData()*.

$scope.nextPage()

Descrizione
Aggiunge allo $scope la funzionalità di passaggio alla pagina di visualizzazione successiva alla corrente. Aumenta di 1 l'indicatore della pagina corrente e esegue una nuova richiesta al server per i dati corrispondenti alla pagina successiva con *getData()*.

$scope.toPage(index)

Parametro

index

Intero rappresentante il numero della pagina desiderata.

Descrizione

Aggiunge allo $scope la funzionalità di passaggio alla pagina di visualizzazione specificata dal parametro *index*. Aggiorna l'indicatore della pagina corrente settandolo all'indice passato per parametro alla funzione. Successivamente esegue una richiesta al server per ottenere i dati per la nuova visualizzazione.

changeSort()

Descrizione

Cambia la variabile $current_sort dello $scope che regola l'ordine di visualizzazione dei Document, da discendente ad ascendente e viceversa.

$scope.columnSort(index)

Parametro

index

Numero intero che rappresenta l'indice della colonna da ordinare.

Descrizione

Funzione che cambia l'ordinamento di visualizzazione di una colonna.
Se la colonna in questione è una colonna diversa da quella che precedentemente ordinata, la variabile *current_sorted_column* dello $scope viene aggiornata, altrimenti la funzione cambia solamente l'ordinamento della colonna.

$scope.deleteDocument(index)

Parametro

index

Indice che rappresenta il Document da eliminare.

Descrizione

Funzione che elimina il Document identificato dall'indice *index*. Effettua una richiesta REST di rimozione alla risorsa pubblicizzata da QueryService. Se l'eliminazione ha successo viene lanciato l'aggiornamento della vista corrente con il Document eliminato, altrimenti si richiama la callback di errore.

5.2.1.11 @RegisterController

Descrizione

Modulo che descrive il Controller della pagina di registrazione.

Utilizzo

Controller utilizzato da AngularJS per fornire e gestire i dati durante la registrazione.

Dipendendenze iniettate al controller

- $scope;
- $location;
- RegisterService;

Attributi

- $scope.credentials;

Funzionalità

signupForm()

Descrizione

Nel caso in cui il form di registrazione sia compilato con dati validi, richiama la richiesta RESTful *query* sulla $resource fornita da RegisterService. Con questa funzione viene creato un nuovo utente sul server in base ai dati inseriti.

5.2.1.12 @UsersCollectionController

Descrizione

Modulo che descrive il Controller della Collection degli user.

Utilizzo

Controller utilizzato da AngularJS per fornire e gestire i dati della Collection degli user.

Dipendendenze iniettate al controller

- $scope;
- $route;
- $location;
- UserCollectionService;
- UserEditService;

Attributi

- $scope.current_sorted_column;
- $scope.column_original_name;
- $scope.current_sort;
- $scope.current_page;
- $scope.current_collection;
- $scope.rows;

Funzionalità

init()

Descrizione

Funzione di inizializzazione del CollectionController. Imposta i valori iniziali delle variabili dello $scope associate alla Collection view ed esegue la funzione *getData()*.

getData()

Descrizione

Richiama la richiesta RESTful *query* sulla $resource fornita da UserCollectionService. Con questa funzione vengono prelevati i dati legati alla Collection da visualizzare e vengono passati alla Collection view tramite il binding con lo $scope.

$scope.numerify(num)

Parametro

num

Numero intero che rappresenta il numero di pagine disponibili.

Descrizione

Funzione ausiliaria che crea un Array di dimensione *num* che verrà poi usato dalla directive ng-repeat di angular per la creazione del numero di pagine disponibili.

$scope.previousPage()

Descrizione

Aggiunge allo $scope la funzionalità di passaggio alla pagina di visualizzazione precedente alla corrente. Diminuisce di 1 l'indicatore della pagina corrente e esegue una nuova richiesta al server per i dati corrispondenti alla pagina precedente con *getData()*.

$scope.nextPage()

Descrizione

Aggiunge allo $scope la funzionalità di passaggio alla pagina di visualizzazione successiva alla corrente. Aumenta di 1 l'indicatore della pagina corrente e esegue una nuova richiesta al server per i dati corrispondenti alla pagina successiva con *getData()*.

$scope.toPage(index)

Parametro

index

Numero intero che rappresenta la pagina desiderata.

Descrizione

Aggiunge la funzionalità di passaggio alla pagina di visualizzazione specificata dal parametro *index*. Aggiorna l'indicatore della pagina corrente settandolo all'indice passato per parametro alla funzione. Successivamente esegue una richiesta al server per ottenere i dati per la nuova visualizzazione.

changeSort()

Descrizione

Cambia la variabile $current_sort dello $scope che regola l'ordine di visualizzazione dei Document, da discendente ad ascendente e viceversa.

$scope.columnSort(index)

Parametro

index

Numero intero che rappresenta la colonna da ordinare.

Descrizione

Funzione che cambia l'ordinamento di visualizzazione di una colonna.
Se la colonna in questione è una colonna diversa da quella che precedentemente ordinata, la variabile *current_sorted_column* dello $scope viene aggiornata, altrimenti la funzione cambia solamente l'ordinamento della colonna.

$scope.deleteDocument(index)

Parametro

index

Indice che rappresenta il Document da eliminare.

Descrizione

Funzione che elimina il profilo identificato dall'indice *index*. Effettua una richiesta REST di rimozione alla risorsa pubblicizzata da *@UserEditService* fornendo come argomenti l'identificativo della Collection correntemente visualizzata e l'indice del Document da eliminare.
Se l'eliminazione ha successo viene lanciato l'aggiornamento della vista corrente con il Document eliminato, altrimenti si richiama la callback di errore.

5.2.1.13 @UsersController

Descrizione

Modulo che descrive il Controller della pagina di visualizzazione di un profilo utente da parte di un amministratore.

Utilizzo

Controller utilizzato da AngularJS per fornire e gestire i dati durante la visualizzazione di un document.

Dipendendenze iniettate al controller

- $scope;
- $location;
- $routeParams;
- UserDataService;
- UserEditService.

Attributi

- $scope.current_document;
- $scope.values;

Funzionalità

UserDataService.query()

Parametro

$scope.current_document

Indicatore del Document corrente.

Descrizione

Richiama la richiesta RESTful *query* sulla $resource fornita da *@UserDataService*. Con questa funzione vengono prelevati i dati legati al document da visualizzare e caricati sullo scope.

$scope.deleteDocument(index)

Parametro

index

Indice del profilo da eliminare.

Descrizione

Funzione che elimina il profilo identificato dall'indice *index*. Effettua una richiesta REST di rimozione alla risorsa pubblicizzata da UserEditService fornendo come argomenti l'identificativo della Collection correntemente visualizzata e l'indice del Document da eliminare.
Se l'eliminazione ha successo viene lanciato l'aggiornamento della vista corrente con il Document eliminato, altrimenti si richiama la callback di errore.

5.2.1.14 @UsersEditController

Descrizione
Modulo che descrive il Controller della pagina di modifica di un profilo.

Utilizzo
Controller utilizzato da AngularJS per fornire e gestire i dati durante la modifica di un profilo.

Dipendendenze iniettate al controller

- $scope;
- $location
- UserEditService;
- $routeParams.

Attributi

- $scope.current_document;
- $scope.original_data;
- $scope.original_keys;

Funzionalità

UserEditService.query()

Parametro

$routeParams.user_id
Campo di $routeParams contenente il codice identificativo dell'utente di cui richiedere il Document.

Descrizione
Richiama la richiesta RESTful *query* sulla $resource fornita da UserEditService. Con questa funzione vengono prelevati i dati legati alla Collection da visualizzare e vengono passati alla Collection view tramite il binding con lo $scope.

edit_document()

Descrizione
Funzione che invia al server il documento modificato nel client. Assembla un JSON a partire da original_keys e original_data. Quest'ultima variabile nel frattempo è stata modificata con i nuovi valori del documento.

delete_document()

Descrizione
Funzione che elimina il profilo identificato dall'indice *index*. Effettua una richiesta REST di rimozione alla risorsa pubblicizzata da UserEditService fornendo come argomenti l'identificativo della Collection correntemente visualizzata e l'indice del Document da eliminare.
Se l'eliminazione ha successo viene lanciato l'aggiornamento della vista corrente con il Document eliminato, altrimenti si richiama la callback di errore.

5.2.2 ::Directives

5.2.2.1 @Passwrod_check

Dipendenze iniettate

- scope;
- elem;
- attrs;
- ctrl.

Descrizione
Directive utilizzata per verificare che il contenuto di due campi di un form sia uguale. Viene utilizzata per verificare che l'utente inserisca correttamente entrambe le password durante la registrazione al fine di evitare richieste inutili al server e per prevenire errori piuttosto che correggerli. Se le due password non sono uguali il form non viene validato e non può essere inviato al server.

5.2.2.2 @Unique

Dipendenze iniettate

- $http;
- $timeout.

Descrizione
Directive utilizzata per verificare che la mail inserita dall'utente durante la registrazione non sia già utilizzata. Se la mail è gia in utilizzo non viene validato il form e non si può procedere alla registrazione.

5.3 ::ModelClient

5.3.1 ::Services

I Servizi del Client sono i tramite tra il Client ed il Server. I componenti di questa sezione sono gli esecutori materiali delle richieste REST per ottenere le informazioni dei dati presenti nei database di analisi e utenti presenti nel Server. Essi pubblicizzano delle risorse su cui i Controller associati manipolano le informazioni richieste dal Client.

5.3.1.1 @AuthService

Descrizione
Modulo che espone un oggetto $resource legato all'autenticazaione utente.

Utilizzo
Viene utilizzato per astrarre il processo di verifica delle credenziali eseguito dal server. Questo modulo esegue una richiesta alle API del server per stabilire se le credenziali inserite dall'utente sono valide all'autenticazione. Il modulo ritorna una $resource per rendere più semplice ed immediato il dialogo tra il Client e il sistema di autenticazione.
La $resource esportata da questo modulo viene utilizzata nel LoginCtrl per verificare le credenziali inserite lato Client.

Funzionalità
Dipendenze iniettate:

- $resource.

L'istruzione di ritorno crea una nuova $resource che espone il metodo *login* che verrà invocato dal LoginCtrl.

5.3.1.2 @CollectionDataService

Descrizione
Modulo che espone un oggetto $resource legato ad una particolare Collection.

Utilizzo
Viene utilizzato come astrazione della risorsa Collection, su cui il CollectionController invocherà delle richieste RESTful aggiungendo i parametri necessari. La risorsa fornita conterrá i Document ad essa correlati e quest'ultimi verranno visualizzati in base alle opzioni della richiesta RESTful.

Funzionalità
Dipendenze iniettate:

- $resource.

L'istruzione di ritorno crea una nuova $resource che modella il comportamento della Collection che verrà individuata tramite l'attributo *col_id*.

5.3.1.3 @CollectionListService

Descrizione
Modulo che espone un oggetto $resource legato alla lista delle Collection disponibili.

Utilizzo
Viene creata una $resource che modella ad alto livello la lista delle Collection disponibili, per essere utilizzata nel DashboardController.

Funzionalità
Dipendenze iniettate:

- $resource.

L'istruzione di ritorno crea una $resource che contiene la lista delle Collection disponibili, con cui interagire tramite richieste RESTful.

5.3.1.4 @DocumentDataService

Descrizione
Modulo che espone un oggetto $resource legato ad un particolare Document.

Utilizzo
Questo servizio ritorna un oggetto $resource legato ad un particolare Document, che verrà utilizzato dal DocumentController tramite richieste RESTful per visualizzarne gli attributi.

Funzionalità
Dipendenze iniettate:

- $resource.

L'oggetto ritornato da questa Factory lega la $resource ritornata al Document *doc_id* della Collection *col_id*. Inoltre, stabilisce che le richieste *query* vengano effettuate tramite metodo GET.

5.3.1.5 @DocumentEditservice

Descrizione
Modulo che espone un oggetto $resource legato ad un particolare Document modificabile.

Utilizzo
Questo servizio ritorna un oggetto $resource legato ad un particolare Document, che verrà utilizzato dal DocumentEditController tramite richieste RESTful, sia per visualizzarne le informazioni, sia per modificarne le proprietà.

Funzionalità
Dipendendenze iniettate:

- $resource.

L'oggetto ritornato da questa Factory lega la $resource ritornata al Document *doc_id* della Collection *col_id*. Inoltre, oltre alla visualizzazione, descrive che le richieste di *update* vengono effettuate con richieste di *PUT*, mentre le richieste di *remove* useranno il metodo http *DELETE*.

5.3.1.6 @IndexService

Descrizione
Modulo che pubblicizza una $resource legata ad un indice.

Utilizzo
Questo servizio ritorna una $resource legata ad un indice fornito dal server, per essere utilizzata dall'IndexCtrl per gestire i vari indici visualizzati nel Client. Il presente modulo viene inoltre iniettato nel QueryCtrl per creare un nuovo indice a partire da una query data. Questa risorsa esportata è dotata di tre metodi di accesso e modifica dei dati dell'indice, ovvero *query*, *insert* e *remove*.

Funzionalità
Dipendendenze iniettate:

- $resource.

La $resource esportata fornisce all'IndexCtrl del Client una modo diretto per interagire con un indice fornito dal server. Questa risorsa dispone di un metodo *query* per eseguire una GET RESTful per ottenere una lista degli indici presenti sul Server. Il metodo *insert* crea un nuovo indice a partire da una delle query più utilizzate. *remove* invece rimuove un indice dalla lista degli indici presenti sul server.

5.3.1.7 @LogoutService

Descrizione
Modulo che fornisce la $resource legata al servizio di logout di un utente autenticato del Client.

Utilizzo
Questo servizio fornisce un'astrazione del metodo di logout eseguito dal server. La $resource esportata dal servizio viene utilizzata nel NavBarCtrl per effettuare il logout di un autente autenticato, tramite il metodo pubblicizzato *logout*.

Funzionalità
Dipendendenze iniettate:

- $resource.

La risorsa ritornata dal modulo astrae la connessione con il server quando si effettua il logout dal sistema. La funzione *logout* associata alla $resource ritornata esegue una richiesta RESTful GET al Server per disconnettere l'utente che ha richiesto l'operazione.

5.3.1.8 @ProfileDataService

Descrizione
Il servizio ritorna una $resource legata ad un particolare profilo utente presente nel Server.

Utilizzo
La risorsa ritornata da questo modulo viene utilizzata dal ProfileCtrl per dialogare con le informazioni associate ad un singolo utente. Il metodo *query* associato alla risorsa serve per ottenere i dati associati al profilo utente associato.

Funzionalità
Dipendendenze iniettate:

- $resource.

La funzione *query* esegue una richiesta GET RESTful al server per ottenere le informazioni di un utente. Tali informazioni vengono poi passate al ProfileController per essere fornite in visualizzazione al Client che le ha richieste.

5.3.1.9 @ProfileEditService

Descrizione
Il servizio ritorna una $resource modificabile legata ad un particolare profilo utente presente nel Server.

Utilizzo
La risorsa ritornata da questo modulo viene utilizzata dal ProfileEditCtrl per dialogare con le informazioni associate ad un singolo utente. Alla risorsa ritornata vengono associati tre metodi: *query*, *update* e *remove*.

Funzionalità
Dipendendenze iniettate:

- $resource.

La funzione *query* esegue una richiesta GET RESTful al server per ottenere le informazioni di un utente. *update* invia nuove informazioni al Server che andranno a sovrascrivere quelle già presenti per aggiornarle. Il metodo *remove* invece richiede al Server di eseguire l'eliminazione di un dato profilo utente. Lato Server viene effettuato un controllo per verificare se l'utente che ha richiesto l'eliminazione ha le credenziali per eliminare u utente. Nel caso in cui il controllo abbia esito positivo, il profilo viene eliminato dal Server.

5.3.1.10 @QueryService

Descrizione
Modulo che esporta una $resource legata ad una query effettuabile sul database di analisi.

Utilizzo
La risorsa ritornata da questo servizio viene utilizzata nel QueryCtrl. Il suddetto controller utilizza la risorsa per richiedere l'esecuzione di una query al Server inviando come argomenti i dati presenti nello $scope in quel momento.
Il modulo associa alla risorsa ritornata le funzioni *query* e *remove*.

Funzionalità
Dipendendenze iniettate:

- $resource.

La funzione *query* esegue una richiesta GET REST al Server per richiedere l'esecuzione di una query sui dati del database di analisi. I parametri su cui effettuare la query sono inviati dal Client sulla base delle condizioni immesse dall'utente e memorizzate nello $scope. La funzione *remove*, invece, richiede al Server l'eliminazione del Document individuato dall'indice *index* presente nello $scope.

5.3.1.11 @RegisterService

Descrizione
Servizio che ritorna una $resource legata ad un profilo utente intento ad effettuare la registrazione.

Utilizzo
La risorsa ritornata da questo servizio viene utilizzata come tramite tra il Client e il Server per gestire la registrazione di un nuovo utente. La risorsa viene utilizzata dal RegisterCtrl per richiedere la registrazione di un nuovo utente. Viene infatti esposto il metodo *register* che consente di inviare una richiesta di registrazione al Server con le credenziali inserite nell'apposito form del Client.

Funzionalità
Dipendendenze iniettate:

- $resource.

La funzione *register* esportata con la risorsa esegue una richiesta POST REST al Server inviando con essa le credenziali dell'utente attualmente presenti nello $scope.

5.3.1.12 @UserCollectionService

Descrizione
Servizio che fornisce una $resource legata alla lista degli utenti ordinari non amministratori.

Utilizzo
La risorsa messa a disposizione dallo UserCollectionService contiene la lista di tutti gli utenti del sistema. Questa $resource viene utilizzata dallo UsersCollectionController per visualizzare correttamente tutti gli utenti del sistema ed organizzarli nell'apposita vista. Il modulo esporta con la $resource il metodo *query*.

Funzionalità
Dipendendenze iniettate:

- $resource.

La funzione *query* effettua una richiesta REST GET al Server per ottenere la lista degli utenti.

5.3.1.13 @UserDataService

Descrizione
Servizio che fornisce una $resource legata ad un particolare utente non amministratore registrato al sistema.

Utilizzo
Questa risorsa viene utilizzata nello UserCtrl per richiedere al Server le informazioni associate ad un utente. Ogni utente è identificato dal proprio *id* e questo campo viene passato dal Client per individuare un singolo utente, se presente, nel Server. La risorsa viene esportata con il metodo *query*, per ottenere i dati voluti.

Funzionalità
Dipendendenze iniettate:

- $resource.

Il metodo *query* esegue una richiesta GET REST al Server richiedendo le informazioni di un utente specifico, individuato da un identificativo univoco, inviato dal Client come argomento della richiesta.

5.3.1.14 @UserEditService

Descrizione
Servizio che fornisce una $resource legata ad un particolare utente con campi modificabili registrato al sistema.

Utilizzo
Questa risorsa viene utilizzata nello UserEditCtrl per richiedere al Server le informazioni associate ad un utente. L'argomento della richiesta al Server è l dell'utente. Le funzioni esportate con la $resource sono: *query*, *update* e *remove*.

Funzionalità
Dipendendenze iniettate:

- $resource.

L'oggetto ritornato da questo servizio lega la $resource ritornata all'utente con il campo *id* uguale a quello utilizzato per la ricerca. Inoltre, oltre alla visualizzazione espressa tramite il metodo *query* realizzato con una richiesta GET RESTful, descrive che le richieste di *update* vengano effettuate con richieste di *PUT*, mentre le richieste di *remove* useranno il metodo http *DELETE*.

6 Diagrammi di sequenza

6.1 Modifica della View con successo

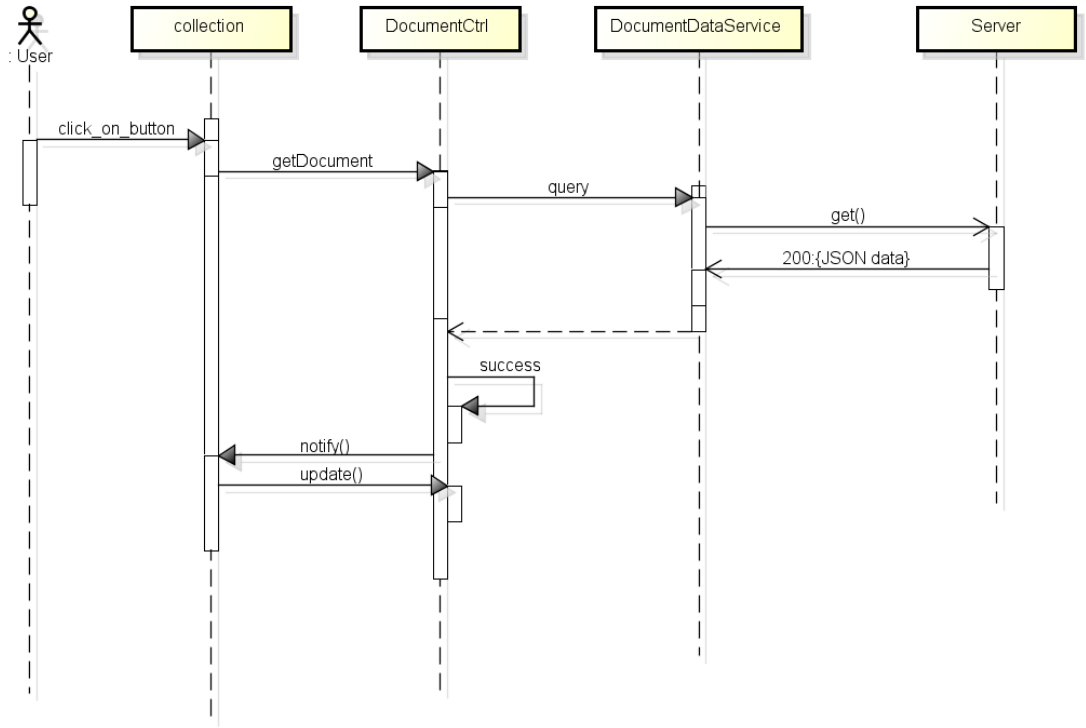

Figura 1: Diagramma sequenza: Modifica View con successo

Il diagramma precedente illustra la sequenza di operazioni che avviene alla richiesta da parte di un utente di visualizzare un document. Dopo aver cliccato il bottone di visualizzazione del document avviene una serie di richieste a cascata fino ad arrivare al server. Quest'ultimo verifica l'autenticità della richiesta e risponde con il documento cercato. Questo documento viene ritornato dal servizio al controller di gestione dei documenti che esegue la callback success e prepara il json ad essere visualizzato. La view viene poi aggiornata con i nuovi dati.

6.2 Modifica della View con insuccesso

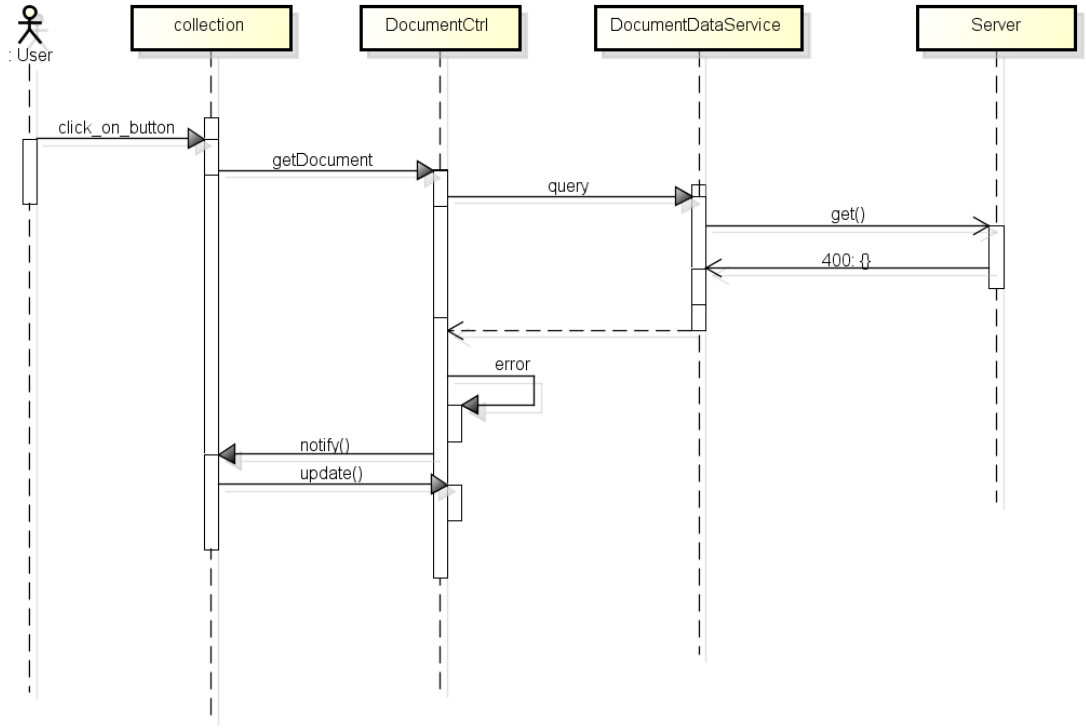

Figura 2: Diagramma sequenza: Modifica View con insuccesso

Questo diagramma illustra la stessa operazione del precedente con esito negativo. Il server rifiuta la richiesta e risponde con un codice di errore. Il controller esegue la callback error e la view verrà aggiornata con un messaggio di errore.