Definizione di Prodotto

Gruppo SteakHolders — Progetto MaaP

Informazioni sul documento

Versione	4.0.0
Redazione	Luca De Franceschi
	Giacomo Fornari
	Serena Girardi
	Enrico Rotundo
	Nicolò Tresoldi
Verifica	Federico Poli, Serena Girardi
Approvazione	Enrico Rotundo
Uso	Esterno
Distribuzione	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Gruppo SteakHolders
	CoffeeStrap

Descrizione

Questo documento descrive la progettazione di dettaglio definita dal gruppo SteakHolders relativa al progetto MaaP.

Registro delle modifiche

Versione	**Data**	**Persone coinvolte**	**Descrizione**
4.0.0	2014-02-25	Enrico Rotundo (Responsabile)	Approvazione.
3.3.0	2014-02-24	Serena Girardi (Verificatore)	Verifica finale.
3.2.2	2014-02-19	Giacomo Fornari (Progettista)	Correzione e incremento.
3.2.1	2014-02-19	Nicolò Tresoldi (Progettista)	Correzione e incremento.
3.2.0	2014-02-19	Luca De Franceschi (Verificatore)	Verifica classi Back-end e tracciamento test.
3.1.0	2014-02-18	Federico Poli (Verificatore)	Verifica classi Front-end e tracciamento test.
3.0.9	2014-02-17	Gianluca Donato (Progettista)	Stesura Tracciamento test di unità.
3.0.8	2014-02-16	Nicolò Tresoldi (Progettista)	Definizione classi Front-end.
3.0.7	2014-02-15	Serena Girardi (Progettista)	Definizione classi Back-end.
3.0.6	2014-02-15	Enrico Rotundo (Progettista)	Stesura Tracciamento test di unità.
3.0.5	2014-02-14	Gianluca Donato (Progettista)	Definizione classi Front-end.
3.0.4	2014-02-14	Giacomo Fornari (Progettista)	Stesura Tracciamento classi-requisiti.
3.0.3	2014-02-14	Serena Girardi (Progettista)	Definizione classi Back-end.
3.0.2	2014-02-13	Nicolò Tresoldi (Progettista)	Stesura Standard di progetto.
3.0.1	2014-02-13	Giacomo Fornari (Progettista)	Creato documento e stesa introduzione.

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Il seguente documento vuole descrivere la progettazione di dettaglio definita per il progetto MaaP. Il documento si basa sulla *Specifica Tecnica v4.0.0*. I *programmatori* si serviranno di tale documento per procedere con le attività di codifica.

1.2 Scopo del prodotto

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

1.3 Glossario

Ogni occorrenza di termini tecnici, di dominio e gli acronimi sono marcati con una "G" in pedice e riportati nel documento *Glossario v4.0.0*.

1.4 Riferimenti

Vengono elencanti i riferimenti su cui si è basata la definizione della progettazione di dettaglio.

1.4.1 Normativi

- **Norme di Progetto:** *Norme di Progetto v4.0.0*;
- **Capitolato d'appalto C1:**
  http://www.math.unipd.it/~tullio/IS-1/2013/Progetto/C1.pdf;
- **Specifica Tecnica:** *Specifica Tecnica v4.0.0*.

2 Standard di progetto

2.1 Standard di progettazione architetturale

Gli standard di progettazione sono definiti nella *Specifica Tecnica v4.0.0*.

Per chiarezza, si evidenzia che in aggiunta al formalismo UML_G 2.0 è stata utilizzata una no: tazione ad hoc per rappresentare il tipo di dato di una funzione: "`function(<parametri>)`" rappresenta quindi il tipo di dato di una funzione che richiede i parametri "`<parametri>`".

2.2 Standard di documentazione del codice

Gli standard per la scrittura della documentazione del codice sono definiti nelle *Norme di Progetto v4.0.0*.

2.3 Standard di denominazione di entità e relazioni

Tutti gli elementi definiti come $package_G$, classi, metodi o attributi, devono avere denominazioni chiare ed esplicative. Il nome deve avere una lunghezza tale da non pregiudicarne la leggibilità e chiarezza. È preferibile utilizzare dei sostantivi per le entità e dei verbi per le relazioni. Le abbreviazioni sono ammesse se:

- immediatamente comprensibili;
- non ambigue;
- sufficientemente contestualizzate.

Le regole tipografiche relative ai nomi delle entità sono definite nelle *Norme di Progetto v4.0.0*.

2.4 Standard di programmazione

Gli standard di programmazione sono definiti e descritti nelle *Norme di Progetto v4.0.0*.

2.5 Strumenti di lavoro

Per gli strumenti di lavoro da utilizzare durante la codifica e le procedure per il loro corretto funzionamento e coordinamento si rimanda al documento *Norme di Progetto v4.0.0*.

3 Specifica classi del Back-end

3.1 Componente Back-end::DeveloperProject

3.1.1 Classe ProjectApp

ProjectApp
+start()

Tabella 2: Classe ProjectApp

Descrizione

Classe modificabile dall'utente-developer che si occupa di configurare e avviare il server dell'applicazione.

Utilizzo

Internamente avvia il server utilizzando la classe ServerApp, a cui passa i parametri di configurazione del progetto definiti con un oggetto della classe ProjectConfig.

Relazioni con altre classi

Utilizza le classi:

- `Back-end::Lib::ServerApp`
- `Back-end::DeveloperProject::Config::ProjectConfig`

Attributi

Assenti

Metodi

`+start()`
Questo metodo statico carica l'oggetto di configurazione di tipo ProjectConfig e fa partire il server dell'applicazione, utilizzando la classe ServerApp del package Lib.

3.1.2 Classe ProjectConfig

ProjectConfig

Tabella 3: Classe ProjectConfig

Descrizione

Questa classe rappresenta la configurazione di un'applicazione.

Utilizzo

Viene passato come parametro al costruttore della classe ServerApp per configurare l'applicazione.

Relazioni con altre classi

Estende la classe:

- `Back-end::Lib::Config`

Attributi

Assenti

Metodi

Assenti

3.2 Componente Back-end::Lib

3.2.1 Classe ServerApp

ServerApp
+start() +ServerApp(config:Config)

Tabella 4: Classe ServerApp

Descrizione

Classe che si occupa di avviare il server e di invocare il *middleware*$_G$. È il componente client del *Design Pattern*$_G$ *Chain of responsibility*$_G$. Utilizza i pacchetti Mongoose ed Express.

Utilizzo

Viene utilizzato per avviare l'applicazione. Internamente inizializza la catena gestione delle chiamate utilizzando la classe `Back-end::Lib::Middleware::MiddlewareLoader`.

Relazioni con altre classi

Utilizza le classi:

- `Back-end::Lib::Controller::Middleware::MiddlewareLoader`
- `Back-end::Lib::Config`

Attributi

Assenti

Metodi

`+start()`
Questo metodo fa partire il server. Non ritorna il controllo finché il server è in funzione.

`+ServerApp(config:Config)`
Questo metodo statico accetta come parametro l'oggetto di configurazione dell'applica: zione e costruisce l'oggetto ServerApp che rappresenta il server dell'applicazione.

- `config:Config`
  È l'oggetto di configurazione dell'applicazione.

3.2.2 Classe Config

Config
+getEnvironment():String +getServerPort():Integer +getServerStaticPath():String +getUserDbUri():String +getDataDbUri():String +getSmtpService():String +getSmtpAuth():String +getDSLPath()

Tabella 5: Classe Config

Descrizione

Classe che rappresenta l'interfaccia della classe di configurazione dell'applicazione.

Utilizzo

Viene utilizzata per descrivere tutti i parametri dell'applicazione. Quando viene creata una ServerApp le viene passato un oggetto di questo tipo ed essa avvierà l'applicazione a partire da questa configurazione.

Relazioni con altre classi

È estesa dalle classi:

- `Back-end::DeveloperProject::Config::ProjectConfig`

Attributi

Assenti

Metodi

`+getEnvironment():String`
Restituisce la variabile d'ambiente che informa se l'applicazione deve essere eseguita in modalità "developing" o "production".

`+getServerPort():Integer`
Restituisce la porta su cui il server deve mettersi in ascolto.

`+getServerStaticPath():String`
Restituisce il percorso della cartella che il server deve utilizzare per fornire file statici.

`+getUserDbUri():String`
Restituisce l'uri del database che il server deve utilizzare come database degli utenti

`+getDataDbUri():String`
Restituisce l'uri del database che il server deve utilizzare come database di analisi, cioè quello contenente le collection di cui l'applicazione deve permettere la visualizzazione.

`+getSmtpService():String`
Restituisce il nome del servizio che potrà essere usato dall'applicazione per inviare email.

`+getSmtpAuth():String`
Restituisce le credenziali con cui è possibile utilizzare il servizio smtp per inviare email.

`+getDSLPath()`
Restituisce la path da cui caricare i file DSL definiti dallo sviluppatore.

3.3 Componente Back-end::Lib::View

3.3.1 Classe ForgotMailView

ForgotMailView
`+buildForgotMail(userMail:String, senderMail:String, tokenlink:String):Email`

Tabella 6: Classe ForgotMailView

Descrizione

Classe che fornisce una rappresentazione della mail.

Utilizzo

Viene utilizzata come template di email da inviare nel caso in cui l'utente richieda il recupero password.

Relazioni con altre classi Assenti

Attributi

Assenti

Metodi

`+buildForgotMail(userMail:String, senderMail:String, tokenlink:String):Email`
Metodo che definisce e restituisce la struttura dell'email da inviare per il reset della password nel formato Email della libreria NodeMailer.

- `userMail:String`
  Parametro che rappresenta l'email dell'utente a cui inviare l'email.
- `senderMail:String`
  Parametro che rappresenta l'email del mittente.
- `tokenlink:String`
  Questo parametro è il link con il token dal quale l'utente può accedere per procedere con il reset della password.

3.4 Componente Back-end::Lib::Controller::Middleware

3.4.1 Classe MiddlewareLoader

MiddlewareLoader
`+init(app:ServerApp)`

Tabella 7: Classe MiddlewareLoader

Descrizione

Classe che definisce un'interfaccia comune per tutte le richieste dell'applicazione. È la componente facade del *Design Pattern*$_G$ *Facade*$_G$ e handler del *Design Pattern*$_G$ *Chain of responsibility*$_G$.

Utilizzo

Viene utilizzato per istanziare in modo nascosto all'applicazione tutti i *middleware*$_G$ presenti nel componente `Back-end::Lib::Middleware`.

Relazioni con altre classi

Utilizza le classi:

- `Back-end::Lib::Controller::Middleware::Router`
- `Back-end::Lib::Controller::Middleware::NotFoundHandler`
- `Back-end::Lib::Controller::Middleware::DSLLoaderHandler`
- `Back-end::Lib::Controller::Middleware::ErrorHandler`

Attributi

Assenti

Metodi

`+init(app:ServerApp)`
Metodo che inserisce in ogni richiesta un riferimento all'applicazione rendendolo accedibile tramite /codereq.app.
Inizializza tutti i middleware richiamando i corrispondenti metodi init.

- `app:ServerApp`
  È l'istanza del server dell'applicazione.

3.4.2 Classe Authentication

Authentication

`+handler(req:Request, res:Response, next:function(MaapError))`
`+init(app:ServerApp)`
`+authenticate(req:Request, res:Response, next:function(MaapError))`
`+requireNotLogged(req:Request, res:Response,`
`next:function(MaapError))`
`+requireLogged(req:Request, res:Response,`
`next:function(MaapError))`
`+requireAdmin(req:Request, res:Response, next:function(MaapError))`
`+requireSuperAdmin(req:Request, res:Response,`
`next:function(MaapError))`

Tabella 8: Classe Authentication

Descrizione

Classe che si occupa dell'autenticazione di un'utente. È uno dei componenti subsystem class del *Design Pattern*$_G$ *Facade*$_G$ e handler del *Design Pattern*$_G$ *Chain of responsibility*$_G$.

Utilizzo

Viene utilizzata per verificare i dati inseriti dall'utente nella pagina di login e controllare l'effettiva corrispondenza delle credenziali nel *database*$_G$.

Relazioni con altre classi

Utilizza le classi:

- `Back-end::Lib::Model::UserModel`

Attributi

Assenti

Metodi

`+handler(req:Request, res:Response, next:function(MaapError))`
Metodo che implementa la gestione delle richieste arrivate da Express: effettuata l'elaborazione passa il controllo al successivo middleware, utilizzando il pattern *Chain of responsibility*$_G$.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.

- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

`+init(app:ServerApp)`
Configura Passport dandogli la strategia che deve utilizzare per l'autenticazione degli utenti e definendo i campi da serializzare e deserializzare per il mantenimento delle informazioni sulla sessione utente.

- `app:ServerApp`
  È l'istanza del server dell'applicazione.

`+authenticate(req:Request, res:Response, next:function(MaapError))`
Utilizza il metodo `authenticate()` di Passport per effettuare l'autenticazione dell' utente.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

`+requireNotLogged(req:Request, res:Response, next:function(MaapError))`
Metodo che verifica se l'utente è autenticato, nel caso lo sia risponde con errore mentre nel caso l'utente non sia autenticato chiama il successivo middleware.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

`+requireLogged(req:Request, res:Response, next:function(MaapError))`
Metodo che deve verificare se l'utente è autenticato, richiamando il middleware successivo in caso lo sia mentre deve ritornare un errore in caso contrario.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

`+requireAdmin(req:Request, res:Response, next:function(MaapError))`
Metodo che verifica se l'utente autenticato ha un livello admin richiamando il successivo middleware in caso affermativo altrimenti rispondendo con un errore.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

`+requireSuperAdmin(req:Request, res:Response, next:function(MaapError))`
Metodo che deve verificare se l'utente autenticato ha livello di super admin richiamando in caso positivo il successivo middleware ed in caso negativo rispondere con errore.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

3.4.3 Classe Router

Router
`+handler(req:Request, res:Response, next:function(MaapError))` `+init(app:ServerApp)`

Tabella 9: Classe Router

Descrizione

Classe che si occupa della richiesta di risorse. È uno dei componenti subsystem class del *Design Pattern*$_G$ *Facade*$_G$ e handler del *Design Pattern*$_G$ *Chain of responsibility*$_G$. Ha una relazione con la classe Authentication, poiché fa uso di alcuni metodi per controllare l'autenticazione.

Utilizzo

Si occupa di smistare la richiesta in base all'*URI*$_G$ ricevuto e ad invocare l'opportuno metodo di creazione sulla classe `Back-end::Lib::Controller::ControllerFactory`.

Relazioni con altre classi

Utilizza le classi:

- `Back-end::Lib::Controller::Middleware::Authentication`
- `Back-end::Lib::Controller::Service::ServiceFactory`

Attributi

Assenti

Metodi

`+handler(req:Request, res:Response, next:function(MaapError))`
Metodo che implementa la gestione delle richieste arrivate da Express: effettuata l'elaborazione passa il controllo al successivo middleware, utilizzando il pattern *Chain of responsibility*$_G$.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.

- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

`+init(app:ServerApp)`
Metodo che definisce per ogni richiesta il corrispondente controller che dovrà gestirla, verificando i permessi dell'utente che la richiede utilizzando i metodi del modulo `Authenticate`.

- `app:ServerApp`
  È l'istanza del server dell'applicazione.

3.4.4 Classe NotFoundHandler

NotFoundHandler
`+handler(req:Request, res:Response, next:function(MaapError))`

Tabella 10: Classe NotFoundHandler

Descrizione

Classe che si occupa la gestione dell'errore di pagina non trovata. È uno dei componenti subsystem class del *Design Pattern*$_G$ *Facade*$_G$ e handler del *Design Pattern*$_G$ *Chain of responsibility*$_G$.

Utilizzo

Viene utilizzata per generare una pagina 404 di errore nel caso in cui l'*URI*$_G$ passato non corrisponda ad una risorsa presente nell'applicazione.

Relazioni con altre classi Assenti

Attributi

Assenti

Metodi

`+handler(req:Request, res:Response, next:function(MaapError))`
Metodo che risponde con un errore.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.

- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

3.4.5 Classe DSLLoaderHandler

DSLLoaderHandler

`+browseFileSystem(errback:function(MaapError), root:String, callback:function(Array))`
`+init(app:ServerApp)`
`+DSLLoaderHandler()`

Tabella 11: Classe DSLLoaderHandler

Descrizione

Classe che si occupa di caricare i DSL_G presenti nel sistema. È uno dei componenti subsystem class del *Design Pattern*$_G$ *Facade*$_G$ e handler del *Design Pattern*$_G$ *Chain of responsibility*$_G$.

Utilizzo

Viene utilizzata per caricare i DSL_G delle *Collection*$_G$ all'interno del *database*$_G$.

Relazioni con altre classi

Utilizza le classi:

- `Back-end::Lib::Model::DSLModel::DSLDomain`

Attributi

Assenti

Metodi

`+browseFileSystem(errback:function(MaapError), root:String, callback:function(Array))`
Metodo che restituisce un array di nomi dei file contenuti nella path `root` data. Nel caso di errore risponde con un oggetto `MaapError` contenente le informazioni dell'errore generato.

- `root:String`
  Path contenente i file DSL_G da caricare.
- `callback:function(Array)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione senza errori, dando come parametro l'array contenente i nomi dei file contenuti nella root.
- `errback:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare nel caso in cui nell'elaborazione avviene un errore. Il parametro di tipo MaapError conterrà le informazioni dell'errore mentre sarà null nel caso in cui non vengano generati errori durante l'elaborazione.

`+init(app:ServerApp)`
Metodo che carica i file DSL_G delle $Collection_G$ utilizzando `browseFileSystem()` facendosi restituire un'array di nomi di file. Per ognuno di questi si occupa di caricarlo correttamente utilizzando il metodo `loadDSLFile()` del modulo `DslDomain`. Nel caso ci siano errori nei DSL_G viene richiamato il successivo middleware, attivando la catena di gestione errore.

- `app:ServerApp`
  È l'istanza del server dell'applicazione.

`+DSLLoaderHandler()`
Questo metodo è il costruttore della classe.

3.4.6 Classe ErrorHandler

ErrorHandler
`+handler(err:MaapError, req:Request, res:Response, next:function(MaapError))`

Tabella 12: Classe ErrorHandler

Descrizione

Questa classe gestisce gli errori generati nei precedenti middleware o controller. Invia al client una risposta con stato HTTP 500 (server error) con una descrizione dell'errore nel formato JSON. È uno dei componenti subsystem class del *Design Pattern*$_G$ *Facade*$_G$ e handler del *Design Pattern*$_G$ *Chain of responsibility*$_G$.

Utilizzo

Questo middleware viene utilizzato per ultimo nella catena di gestione delle richieste di Express, in modo da gestire tutti gli errori generati precedentemente.

Relazioni con altre classi Assenti

Attributi

Assenti

Metodi

`+handler(err:MaapError, req:Request, res:Response, next:function(MaapError))`
Gestisce la richiesta rispondendo con un json contenente le informazioni dell'errore.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.
- `err:MaapError`
  Questo oggetto rappresenta l'errore di tipo MaapError arrivato al server che il metodo deve gestire.

3.5 Componente Back-end::Lib::Controller::Service

3.5.1 Classe ServiceFactory

ServiceFactory
+getCollectionController(app:ServerApp) +getProfileController(app:ServerApp) +getAuthController(app:ServerApp) +getForgotController(app:ServerApp) +getUserController(app:ServerApp) +getShowController(app:ServerApp) +getIndexController(app:ServerApp)

Tabella 13: Classe ServiceFactory

Descrizione

Classe che si occupa di istanziare e restituire una classe *Controller*. Rappresenta il componente creator del *Design Pattern*$_G$ *Factory method*$_G$.

Utilizzo

Viene costruita una sola volta dalla classe *Back-end::Lib::Middleware::Router* e si occupa di creare e restituire l'oggetto *Controller* richiesto.

Relazioni con altre classi

Utilizza le classi:

- `Back-end::Lib::Controller::Service::ForgotService`
- `Back-end::Lib::Controller::Service::ProfileService`
- `Back-end::Lib::Controller::Service::UserService`
- `Back-end::Lib::Controller::Service::ShowService`
- `Back-end::Lib::Controller::Service::IndexService`
- `Back-end::Lib::Controller::Service::CollectionListService`

Attributi

Assenti

Metodi

`+getCollectionController(app:ServerApp)`
Ritorna la classe `collectionController`.

- `app:ServerApp`
  È l'istanza del server dell'applicazione.

`+getProfileController(app:ServerApp)`
Metodo che ritorna la classe `profileController`.

- `app:ServerApp`
  È l'istanza del server dell'applicazione.

`+getAuthController(app:ServerApp)`
Metodo ritornante la classe `authController`.

- `app:ServerApp`
  È l'istanza del server dell'applicazione.

`+getForgotController(app:ServerApp)`
Metodo che restituisce la classe `forgotController`.

- `app:ServerApp`
  È l'istanza del server dell'applicazione.

`+getUserController(app:ServerApp)`
Metodo che deve restituire la classe `userController`.

- `app:ServerApp`
  È l'istanza del server dell'applicazione.

`+getShowController(app:ServerApp)`
Metodo che ritorna la classe `showController`.

- `app:ServerApp`
  È l'istanza del server dell'applicazione.

`+getIndexController(app:ServerApp)`
Questo metodo restituisce la classe `indexController`.

- `app:ServerApp`
  È l'istanza del server dell'applicazione.

3.5.2 Classe ForgotService

ForgotService
`+passwordResetRequest(req:Request, res:Response, next:function(MaapError))`

Tabella 14: Classe ForgotService

Descrizione

Classe che rappresenta il sistema di recupero e ripristino password. È uno dei componenti product del *Design Pattern*$_G$ *Factory method*$_G$.

Utilizzo

La classe fornisce dei metodi per effettuare una richiesta di reset password e, in un secondo momento, procedere al suo ripristino. La richiesta di reset avviene mandando un'email all'indirizzo dell'utente tramite la classe `Back-end::Lib::Middleware::Mailer`. All'interno di questo messaggio sarà presente un link che procederà ad effettuare il login dell'utente e a reindirizzarlo nella pagina di modifica profilo, dalla quale potrà modificare la password.

Relazioni con altre classi

Utilizza le classi:

- `Back-end::Lib::View::ForgotMailView`

Attributi

Assenti

Metodi

`+passwordResetRequest(req:Request, res:Response, next:function(MaapError))`
Metodo che si occupa di impostare l'email e di inviarla per il reset della password utente, costruendone il template e creando il link col token associato alla richiesta.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

3.5.3 Classe ProfileService

ProfileService

`+login(req:Request, res:Response, next:function(MaapError))`
`+logout(req:Request, res:Response, next:function(MaapError))`
`+getProfile(req:Request, res:Response, next:function(MaapError))`
`+updatePassword(req:Request, res:Request,`
`next:function(MaapError))`

Tabella 15: Classe ProfileService

Descrizione

Classe che rappresenta la gestione di un profilo utente, il login e il logout. È uno dei componenti product del *Design Pattern*$_G$ *Factory method*$_G$.

Utilizzo

Viene utilizzata per visualizzare il profilo dell'utente, tramite GET, e per editarlo tramite PUT. Viene anche utilizzata per gestire i dati di e le operazioni relativi all'autenticazione utente e al suo logout dall'applicazione, occupandosi della creazione della sessione utente e della sua distruzione tramite *cookies*$_G$.

Relazioni con altre classi Assenti

Attributi

Assenti

Metodi

`+login(req:Request, res:Response, next:function(MaapError))`
Metodo che si occupa di reindirizzare l'utente alla pagina *dashboard*$_G$.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

`+logout(req:Request, res:Response, next:function(MaapError))`
Questo metodo si occupa di distruggere la sessione utente e di reindirizzarlo alla pagina principale dell'applicazione.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

`+getProfile(req:Request, res:Response, next:function(MaapError))`
Metodo che risponde con le informazioni del profilo dell'utente. In caso avvengano errori, il metodo risponde con un json contenente le informazioni relative all'errore.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

`+updatePassword(req:Request, res:Request, next:function(MaapError))`
Questo metodo modifica la password utente servendosi del metodo `updatePassword`

della classe `Back-end::Lib::Model::UserModel` e rispondendo con una stringa in caso di successo mentre in caso di fallimento con un json di errore.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Request`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

3.5.4 Classe UserService

UserService

`+usersList(req:Request, res:Response, next:function(MaapError))`
`+insertUser(req:Request, res:Response, next:function(MaapError))`
`+registerUser(req:Request, res:Response, next:function(MaapError))`
`+userIdShowPage(next:function(MaapError), req:Request, res:Response)`
`+deleteUser(req:Request, res:Response, next:function(MaapError))`
`+updateLevel(req:Request, res:Response, next:function(MaapError))`

Tabella 16: Classe UserService

Descrizione

Classe che si occupa della varie operazioni che l'admin può compiere sugli utenti dell'applicazione. È uno dei componenti product del *Design Pattern*$_G$ *Factory method*$_G$.

Utilizzo

Viene utilizzata per visualizzare la *index-page*$_G$ degli utenti, visualizzare le relative *show-page*$_G$, eliminare un utente e modificare il profilo. Mette a disposizione dei metodi per effettuare tutte queste operazioni.

Relazioni con altre classi Assenti

Attributi

Assenti

Metodi

`+usersList(req:Request, res:Response, next:function(MaapError))`
Metodo che chiama la funzione `getUserList()` dello schema utente in `Back-end::Lib::Model::UserModel` facendosi restituire la lista di tutti gli utenti presenti nel sistema e restituendola in risposta. Nel caso si verifichi un errore risponde invece con un errore.

- **`req:Request`**
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- **`res:Response`**
  Questo oggetto viene modificato dal metodo durante l'elaborazione, rappresenta la risposta che il server dovrà rispondere.
- **`next:function(MaapError)`**
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

`+insertUser(req:Request, res:Response, next:function(MaapError))`
Metodo che Inserisce un nuovo utente nel database. Nel caso l'elaborazione abbia causato errori risponde con un json di informazioni sull'errore.

- **`req:Request`**
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- **`res:Response`**
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- **`next:function(MaapError)`**
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

`+registerUser(req:Request, res:Response, next:function(MaapError))`
Metodo che registra un nuovo utente nel database tramite la funzione `registerUser()` della classe `Back-end::Lib::Model::UserModel`, rispondendo con una stringa in caso di successo o con un json di errore in caso di fallimento dell'elaborazione.

- **`req:Request`**
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- **`res:Response`**
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- **`next:function(MaapError)`**
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

`+userIdShowPage(next:function(MaapError), req:Request, res:Response)`
Metodo che risponde con i dati di un utente ottenuti tramite la funzione `getUserById()` della classe `Back-end::Lib::Model::UserModel`, in caso di errore risponderà con un json di errore.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

`+deleteUser(req:Request, res:Response, next:function(MaapError))`
Questo metodo elimina un utente dal database utenti, utilizzando la funzione predisposta dal modello utente `deleteUser()`. Nel caso si verifichi un errore durante l'esecuzione, il metodo risponde con un json contenente le informazioni dell'errore.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

`+updateLevel(req:Request, res:Response, next:function(MaapError))`
Questo metodo modifica il livello di un utente tramite la funzione `updateLevel()` della classe `Back-end::Lib::Model::UserModel`.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

3.5.5 Classe ShowService

ShowService

`+getShowPage(req:Request, res:Response, next:function(MaapError))`
`+deleteDocument(req:Request, res:Response,`
`next:function(MaapError))`

Tabella 17: Classe ShowService

Descrizione

Classe che si occupa della gestione della risorsa show-page. È uno dei componenti *product* del *Design Pattern*$_G$ *Factory method*$_G$.

Utilizzo

Viene utilizzata per gestire una richiesta della risorsa show-page, delegando alla classe *Back-end::Lib::DSLModel::DSLDomain* il compito di eseguire la query e restituire i dati in formato JSON.

Relazioni con altre classi Assenti

Attributi

Assenti

Metodi

`+getShowPage(req:Request, res:Response, next:function(MaapError))`
Questo metodo si occupa di andare a prelevare il `Back-end::Lib::Model::DSLModel::DSLCollectionModel` eseguendo una ricerca all'interno del registro della classe `Back-end::Lib::Model::DSLModel::DSLDomain`. Se la ricerca ha successo, viene restituita una `DSLCollectionModel`. A questo punto si ottiene lo `showModel` dal `DSLCollectionModel` e da quest'ultimo viene richiesta la rappresentazione della show-page in formato json, il quale viene utilizzato come risposta del server.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza

del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

`+deleteDocument(req:Request, res:Response, next:function(MaapError))`
Questo metodo si occupa di eliminare un *document*$_G$ dalla *Collection*$_G$.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

3.5.6 Classe IndexService

IndexService
`+getIndexPage(req:Request, res:Response, next:function(MaapError))`

Tabella 18: Classe IndexService

Descrizione

Classe di gestione per la risorsa index È uno dei componenti product del *Design Pattern*$_G$ *Factory method*$_G$.

Utilizzo

Viene utilizzata per gestire la risorsa corrispondente all'index-page di un *Document*$_G$, of: frendo metodi per restituirne gli attributi, effettuarne la modifica o la cancellazione e dele: ga la visualizzazione dell'index-page alla classe `Back-end::Lib::DSLModel::DSLDomain`.

Relazioni con altre classi Assenti

Attributi

Assenti

Metodi

`+getIndexPage(req:Request, res:Response, next:function(MaapError))`
Questo metodo si occupa di andare a prelevare il `Back-end::Lib::Model::DSLModel::DSLCollectionModel` eseguendo una ricerca all'interno del registro della classe `Back-end::Lib::Model::DSLModel::DSLDomain`. Se la ricerca ha successo, viene restituita una `DSLCollectionModel`. A questo punto si ottiene l' `indexModel` dal `DSLCollectionModel` e da quest'ultimo viene richiesta la rappresentazione della index-page in formato json, il quale viene utilizzato come risposta del server.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

3.5.7 Classe CollectionListService

CollectionListService
`+list(req:Request, res:Response, next:function(MaapError))` `+getDashboard(req:Request, res:Response, next:function(MaapError))`

Tabella 19: Classe CollectionListService

Descrizione

Classe di gestione per la risorsa Collection. È uno dei componenti product del *Design Pattern*$_G$ *Factory method*$_G$.

Utilizzo

Viene utilizzata per gestire la risorsa corrispondente alle Collection, offrendo metodi per restituire tutte le collection presenti nell'applicazione.

Relazioni con altre classi Assenti

Attributi

Assenti

Metodi

`+list(req:Request, res:Response, next:function(MaapError))`
Questo metodo risponde con la lista dei nomi delle collection presenti nell'applicazione.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

`+getDashboard(req:Request, res:Response, next:function(MaapError))`
Metodo che si occupa di restituire i dati necessari per la visualizzazione della Dashboard.

- `req:Request`
  Questo oggetto rappresenta la richiesta di tipo Request arrivata al server che il metodo deve gestire.
- `res:Response`
  Questo oggetto rappresenta la risposta che il server dovrà rispondere al termine dell'elaborazione.
- `next:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo MaapError attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

3.6 Componente Back-end::Lib::Model

3.6.1 Classe UserModel

UserModel
- UserSchema:Schema
+init(app:ServerApp) +getUserList(callback:function(JSON[], String), errback:function(MaapError)) +createUser(newUser:JSON, callback:function(JSON), errback:function(MaapError)) +registerUser(newUser:JSON, callback:function(JSON), errback:function(MaapError)) +getUserById(callback:function(JSON,String), userId :String, errback:function(MaapError)) +deleteUser(userId :String, callback:function(String), errback:function(MaapError)) +updatePassword(userId :String, callback:function(String), errback:function(MaapError)) +updateLevel(errback:function(MaapError), userId:String, newLevel:String, callback:function(String))

Tabella 20: Classe UserModel

Descrizione

Classe che si occupa dei metodi per la gestione dei dati utente.

Utilizzo

Viene utilizzata per l'interfacciamento con la libreria *Mongoose*$_G$ per la registrazione dello schema dei dati, e con la libreria passport-local-mongoose per il popolamento automatico dello schema con campi dati e metodi predefiniti. Il costruttore del modello dello schema dei dati viene registrato nella *Factory*$_G$ di *Mongoose*$_G$ ed ogni istanza condividerà la stessa connessione al server.

Relazioni con altre classi Assenti

Attributi

`- UserSchema:Schema`
Questo campo dati rappresenta lo schema *Mongoose*$_G$ dell'utente *MaaP*$_G$.
Lo schema prevede tre attributi:

`email` di tipo `String`

`password` di tipo `String`

`level` di tipo `enum` con tre possibili valori:

1. Utente
2. Admin
3. SuperAdmin

Metodi

`+init(app:ServerApp)`
Metodo che definisce lo schema *mongoose*$_G$ dell'utente rendendo disponibili i metodi da utilizzare per la modifica/creazione/eliminazione di quest'ultimo.

- `app:ServerApp`
  È l'istanza del server dell'applicazione.

`+getUserList(callback:function(JSON[], String), errback:function(MaapError))`
Metodo che restituisce la lista degli utenti in formato json.

- `callback:function(JSON[], String)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione senza errori passandogli la lista di utenti e un messaggio.
- `errback:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare a seguito di un errore durante l'elaborazione.

`+createUser(newUser:JSON, callback:function(JSON), errback:function(MaapError))`
Metodo che crea un nuovo utente nel database degli utenti. Al termine dell'operazione senza errori risponde con un json con le informazioni dell'utente appena creato altrimenti risponde con un errore.

- `newUser:JSON`
  Questo parametro rappresenta i dati utente da utilizzare nella creazione di un nuovo utente.
- `callback:function(JSON)`
  Questo parametro rappresenta la callback che il metodo deve chiamare al termine dell'elaborazione senza errori, dandogli il json con le informazioni dell'utente creato.
- `errback:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare se si sono verificati errori durante l'elaborazione passandogli l'errore.

`+registerUser(newUser:JSON, callback:function(JSON), errback:function(MaapError))`
Questo metodo registra un utente nel database utenti.

- `newUser:JSON`
  Parametro che rappresenta le informazioni dell'utente di cui effettuare la registrazione.
- `callback:function(JSON)`
  Parametro che rappresenta la callback che il metodo invoca al termine dell'elaborazione senza errori dandogli come parametro il json contenente le informazioni dell'utente appena registrato.

- `errback:function(MaapError)`
  Parametro rappresentante la callback richiamata se nell'elaborazione avvengono errori.

`+getUserById(callback:function(JSON,String), userId :String, errback:function(MaapError))`
Metodo che ritorna dato un id, le informazioni dell'utente corrispondente.

- `callback:function(JSON,String)`
  Parametro che rappresenta la callback che il metodo al termine dell'elaborazione senza errori dovrà richiamare passando come parametri il json contenente le informazioni dell'utente e un messaggio.
- `errback:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo deve richiamare se nell'elaborazione si sono verificati errori passando come parametro l'errore.
- `userId :String`
  Parametro corrispondente all'id dell'utente di cui si richiedono le informazioni.

`+deleteUser(userId :String, callback:function(String), errback:function(MaapError))`
Questo metodo elimina un utente dal database.

- `userId :String`
  Parametro rappresentante l'id dell'utente da eliminare.
- `callback:function(String)`
  Parametro corrispondente alla callback che il metodo deve chiamare al termine delle operazioni con un messaggio.
- `errback:function(MaapError)`
  Questo parametro è la callback che il metodo deve richiamare al verificarsi di un errore.

`+updatePassword(userId :String, callback:function(String), errback:function(MaapError))`
Questo metodo si occupa di modificare il dato corrispondente alla password di un utente presente nel database delle credenziali utente.

- `userId :String`
  Parametro rappresentante l'id dell'utente di cui modificare i dati.
- `callback:function(String)`
  Parametro che rappresenta la callback chiamata dal metodo al termine dell'elaborazione senza errori.
- `errback:function(MaapError)`
  Parametro che rappresenta la callback chiamata dal metodo al verificarsi di errori.

`+updateLevel(errback:function(MaapError), userId:String, newLevel:String, callback:function(`
Questo metodo modifica il livello di un utente presente nel database degli utenti.

- `userId:String`
  Parametro corrispondente all'id dell'utente il cui livello deve essere modificato.
- `newLevel:String`
  Parametro che rappresenta il dato sul livello utente.
- `callback:function(String)`
  Questo parametro rappresenta la callback che il metodo deve richiamare al termine dell'elaborazione senza errori.

- `errback:function(MaapError)`
  Parametro che rappresenta la callback chiamata dal metodo al verificarsi di errori durante la modifica.

3.7 Componente Back-end::Lib::Model::DSLModel

3.7.1 Classe DSLDomain

DSLDomain
- modelRegistry:Array - errorRegistry:Array
+loadDSLFile(path:String, callback:function(String), errback:function(MaapError)) +registerCollection(name:String, model:DslCollectionModel) +getCollectionModel(collectionName:String, callback:function(DslCollectionModel), errback:function(MaapError)) +getErrors(callback:function(Array)) +DSLDomain()

Tabella 21: Classe DSLDomain

Descrizione

Classe che si occupa di caricare i file DSL_G. Implementa il *Design Pattern*$_G$ *registry*$_G$.

Utilizzo

Viene utilizzata per caricare dinamicamente tutti i DSL_G a partire dal *database*$_G$ che le viene passato.

Relazioni con altre classi

Utilizza le classi:

- `Back-end::Lib::Model::DSLModel::DSLInterpreterStrategy`
- `Back-end::Lib::Model::DSLModel::DSLCollectionModel`

Attributi

`- modelRegistry:Array`
Questo campo dati rappresenta un registro all'interno del quale sono contenuti tutti i `DSLCollectionModel` caricati all'avvio del server.

`- errorRegistry:Array`
Questo campo dati contiene un registro di `MaapError` generati durante il caricamento e l'interpretazione dei file DSL.

Metodi

`+loadDSLFile(path:String, callback:function(String), errback:function(MaapError))`
Questo metodo prende in input il *path* di un file DSL da andare ad interpretare. Per fare ciò legge il contenuto testuale del file e lo converte in stringa. Questa stringa viene poi passata all'interprete del DSL tramite una chiamata. Questa chiamata restituirà tramite una callback un array di `DSLCollectionModel` che andranno inserite nel registro. Se avviene un errore nella lettura del file o nell'interpretazione del DSL viene sollevato un `MaapError` che viene aggiunto al registro degli errori e restituito alla classe chiamante tramite una callback (che in questo contesto sarebbe più corretto chiamare *errback*.

- `path:String`
  Rappresenta il percorso del file da leggere.
- `callback:function(String)`
  Rappresenta la funzione callback da chiamare una volta che il metodo è stato eseguito con successo. Prende come parametro un messaggio.
- `errback:function(MaapError)`
  Rappresenta la funzione callback da chiamare nel caso in cui avvenga un errore nella chiamata del metodo. In questo caso la funzione prende un `MaapError`.

`+registerCollection(name:String, model:DslCollectionModel)`
Questo metodo si occupa di inserire nel registro delle $DSLCollectionModel_G$ un `DSLCollectionModel` e il nome della Collection.

- `name:String`
  Questo parametro rappresenta il nome della Collection.
- `model:DslCollectionModel`
  Questo parametro rappresenta il DslCollectionModel da inserire nel registro.

`+getCollectionModel(collectionName:String, callback:function(DslCollectionModel), errback:function(MaapError))`
Questo metodo effettua una ricerca all'interno del registro in base al nome indicato. Se trova il DslCollectionModel allora lo restituisce tramite una callback e termina correttamente, altrimenti restituisce un MaapError tramite una callback, segnalando che la Collection non è stata trovata.

- `collectionName:String`
  Questo parametro rappresenta il nome della Collection da cercare all'interno del registro.
- `callback:function(DslCollectionModel)`
  Questo parametro rappresenta la funzione callback da chiamare nel caso in cui il metodo venga eseguito correttamente. La callback prende in input il DslCollectionModel da restituire.
- `errback:function(MaapError)`
  Questa callback viene eseguita nel caso in cui il metodo contenga degli errori o sollevi delle eccezioni. Prende in input il MaapError da restituire.

`+getErrors(callback:function(Array))`
Questo metodo restituisce tramite una callback il registro degli errori generati.

- `callback:function(Array)`
  Questa callback prende in input l'array di errori da restituire.

`+DSLDomain()`
È il metodo costruttore della classe.

3.7.2 Classe DSLInterpreterStrategy

DSLInterpreterStrategy
+loadDSLFile()()

Tabella 22: Classe DSLInterpreterStrategy

Descrizione

Classe astratta che definisce l'interfaccia dell'algoritmo di interpretazione del linguaggio DSL_G utilizzato. È il componente strategy del *Design Pattern*$_G$ *strategy*$_G$.

Utilizzo

Viene utilizzata per incapsulare e rendere intercambiabile l'algoritmo di interpretazione del linguaggio DSL_G. In questo modo, se in futuro vi fosse necessità di cambiare l'algoritmo di interpretazione l'algoritmo può variare indipendentemente dal client che ne farà uso.

Relazioni con altre classi

È estesa dalle classi:

- `Back-end::Lib::Model::DSLModel::DSLInterpreterStrategy::ConcreteDSLInterpreter`

Attributi

Assenti

Metodi

`+loadDSLFile()()`
Questo metodo si occupa di interpretare il contenuto del file DSL passatogli, per poi generare ed eseguire il codice derivato dalla trasformazione.

3.7.3 Classe ConcreteDSLInterpreter

ConcreteDSLInterpreter
`- macro:sweet.js`

`+init(callback:function(String), errback:function(MaapError))`
`+loadDSLFile(content:String, callback:function(DslCollectionModel), errback:MaapError)`
`+DSLConcreteStrategy()`

Tabella 23: Classe ConcreteDSLInterpreter

Descrizione

Classe che concretizza l'interprete del DSL_G. È uno dei componenti ConcreteStrategy del *Design Pattern*$_G$ *Strategy*$_G$.

Utilizzo

Viene utilizzata per implementare l'algoritmo utilizzato nell'interfaccia `Back-end::Lib::DSLModel::DSLInter` per l'interpretazione del linguaggio DSL_G. Conterrà al suo interno un metodo che genererà il *parser*$_G$ a partire da una grammatica regolare.

Relazioni con altre classi

Estende la classe:

- `Back-end::Lib::Model::DSLModel::DSLInterpreterStrategy`

Attributi

`- macro:sweet.js`
Si tratta del modulo *sweet.js* che si occupa dell'interpretazione del file DSL tramite l'utilizzo di macro.

Metodi

`+init(callback:function(String), errback:function(MaapError))`
Questo metodo viene invocato per impostare l'interprete nel modo corretto. Viene settata il modulo sweet.js a partire dal codice di definizione delle macro. Nel caso in cui la classe venga settata correttamente viene invocata una callback che segnalerà la corretta impostazione, altrimenti viene sollevata una callback di errore.

- `callback:function(String)`
  Questa callback viene eseguita al termine del settaggio della classe per segnalare che tutto è avvenuto correttamente tramite l'invio di un messaggio.

- `errback:function(MaapError)`
  Questa callback viene eseguita nel caso in cui la classe non venga settata in modo corretto a causa di un fallimento nella lettura del codice di macro. In questo caso viene restituito l'errore tramite questa callback.

`+loadDSLFile(content:String, callback:function(DslCollectionModel), errback:MaapError)`
Questo metodo si occupa di interpretare il contenuto del file DSL passatogli, per poi generare ed eseguire il codice derivato dalla trasformazione tramite le macro di *sweet.js*. Se il codice viene generato ed eseguito correttamente allora avviene una chiamata alla callback di successo, altrimenti avviene una chiamata alla callback di errore.

- `content:String`
  Questo parametro rappresenta il contenuto testuale del file DSL da interpretare.
- `callback:function(DslCollectionModel)`
  Questa callback viene chiamata alla terminazione positiva del metodo e riceve in input l'array di DslCollectionModel da restituire al chiamante.
- `errback:MaapError`
  Questa callback viene eseguita nel caso in cui ci sia un fallimento nell'interpretazione del DSL o nell'esecuzione del codice generato a partire da esso. Prende in input un MaapError da restituire alla funzione chiamante.

`+DSLConcreteStrategy()`
Questo metodo è il costruttore della classe.

3.7.4 Classe DSLCollectionModel

DSLCollectionModel
- showModel:ShowModel - indexModel:IndexModel - collectionName:String
+DSLCollectionModel(showModel:ShowModel, indexModel:IndexModel, collectionName:String) +getCollectionName():String +getIndexModel():IndexModel +getShowModel():ShowModel +setIndexModel(indexModel:IndexModel) +setShowModel(showModel:ShowModel)

Tabella 24: Classe DSLCollectionModel

Descrizione

Classe che si occupa di definire il model della *Collection*$_G$ a partire dal *DSL*$_G$. Si ispira all'*Abstract Syntax Tree*$_G$.

Utilizzo

È l'oggetto risultante dell'interpretazione del DSL_G. Definisce una rappresentazione interna di una $Collection_G$.

Relazioni con altre classi

Utilizza le classi:

- `Back-end::Lib::Model::DSLModel::ShowModel`
- `Back-end::Lib::Model::DSLModel::IndexModel`

Attributi

`- showModel:ShowModel`
Questo campo dati rappresenta lo ShowModel della Collection.

`- indexModel:IndexModel`
Questo campo dati rappresenta l'IndexModel della classe.

`- collectionName:String`
Questo campo dati rappresenta il nome della Collection.

Metodi

`+DSLCollectionModel(showModel:ShowModel, indexModel:IndexModel, collectionName:String)`
Questo metodo è il costruttore pubblico della classe.

- `showModel:ShowModel`
  Questo parametro è un riferimento allo showModel della classe.
- `indexModel:IndexModel`
  Questo parametro è un riferimento all'indexModel della classe.
- `collectionName:String`
  Questo parametro rappresenta il nome della Collection da creare.

`+getCollectionName():String`
Questo metodo restituisce il campo collectionName della classe.

`+getIndexModel():IndexModel`
Questo metodo restituisce il campo indexModel della classe.

`+getShowModel():ShowModel`
Questo metodo restituisce il campo showModel della classe.

`+setIndexModel(indexModel:IndexModel)`
Questo metodo imposta il campo dati showModel della classe con il parametro ricevuto in input.

- `indexModel:IndexModel`
  Questo parametro rappresenta l'IndexModel da settare.

`+setShowModel(showModel:ShowModel)`
Questo metodo si occupa di settare il campo showModel con il parametro ricevuto in input.

- `showModel:ShowModel`
  Questo parametro rappresenta lo ShowModel da settare.

3.7.5 Classe ShowModel

ShowModel
- collectionName:String - DataSchema:Schema - attributes:Array
+ShowModel(collectionName:String) +addAttribute(attribute:Attribute) +getAttributes():Array +getData(collectionName:String, documentId:String, callback:function(JSON), errback:function(MaapError))

Tabella 25: Classe ShowModel

Descrizione

Classe che racchiude tutte le informazioni relative ad una show-page. Tali informazioni vengono dichiarate dal developer nel DSL. È composta da un numero variabile di attributi, definiti dalla classe `Back-end::Lib::DSLModel::Attribute`.

Utilizzo

Questa classe viene creata dalla componente che si occupa di caricare il DSL (interpretandolo o facendone il parsing).

Relazioni con altre classi

Utilizza le classi:

- `Back-end::Lib::Model::DSLModel::Attribute`

Attributi

`- collectionName:String`
Questo campo dati rappresenta il nome della Collection.

`- DataSchema:Schema`
Questo campo dati rappresenta lo schema *Mongoose*$_G$ dei dati da rappresentare.

`- attributes:Array`
Questo campo dati rappresenta l'array di Attribute del modello.

Metodi

`+ShowModel(collectionName:String)`
Questo metodo è il costruttore della classe. Si occupa di definire lo schema e di inzializzare l'array di attributi.

- `collectionName:String`
  Questo parametro rappresenta il nome della Collection.

`+addAttribute(attribute:Attribute)`
Questo metodo si occupa di aggiungere l'*Attribute* indicato nell'array *attributes* della classe.

- `attribute:Attribute`
  Questo parametro rappresenta l'attributo da aggiungere all'array.

`+getAttributes():Array`
Questo metodo si occupa di restituire l'array di attributi del modello.

`+getData(collectionName:String, documentId:String, callback:function(JSON), errback:function(MaapError))`
Questo metodo si occupa di collegarsi a $MongoDB_G$ tramite $Mongoose_G$ e di restituire il JSON con tutti i dati tramite una callback, in modo da poter generare correttamente la show-page. In caso di fallimento il metodo restituisce un errore tramite una callback di errore.

- `collectionName:String`
  Questo parametro rappresenta il nome della Collection sulla quale effettuare l'estrazione da $MongoDB_G$.
- `documentId:String`
  Questo parametro rappresenta il codice identificativo del Document sul quale effettuare l'estrazione dei dati.
- `callback:function(JSON)`
  Questa callback viene chiamata al termine dell'estrazione dei dati e riceve in input il JSON da restituire alla funzione chiamante.
- `errback:function(MaapError)`
  Questa callback viene chiamata nel caso in cui avvenga un errore nell'estrazione dei dati da $MongoDB_G$. Prende in input l'errore da restituire alla funzione chiamante.

3.7.6 Classe IndexModel

IndexModel
- collectionName:String - DataSchema:Schema - attributes:Array
+IndexModel() +addAttribute(attribute:Attribute) +getAttributes():Array +getData(errback:function(MaapError), collectionName:String, callback:function(JSON))

Tabella 26: Classe IndexModel

Descrizione

Classe che racchiude tutte le informazioni relative ad una index-page. Tali informazioni vengono dichiarate dal developer nel DSL. È composta da un numero variabile di attributi, definiti dalla classe `Back-end::Lib::DSLModel::Attribute`.

Utilizzo

Questa classe viene creata dalla componente che si occupa di caricare il DSL (interpretandolo o facendone il parsing).

Relazioni con altre classi Assenti

Attributi

`- collectionName:String`
Questo campo dati rappresenta il nome della Collection.

`- DataSchema:Schema`
Questo campo dati rappresenta lo schema *Mongoose*$_G$ dei dati da rappresentare.

`- attributes:Array`
Questo campo dati rappresenta l'array di Attribute del modello.

Metodi

`+IndexModel()`
Questo metodo è il costruttore della classe. Si occupa di definire lo schema e di inizializzare l'array di attributi.

`+addAttribute(attribute:Attribute)`
Questo metodo si occupa di aggiungere l'*Attribute* indicato nell'array *attributes* della classe.

- `attribute:Attribute`
  Questo parametro rappresenta l'attributo da aggiungere all'array.

`+getAttributes():Array`
Questo metodo si occupa di restituire l'array di attributi del modello.

`+getData(errback:function(MaapError), collectionName:String, callback:function(JSON))`
Questo metodo si occupa di collegarsi a $MongoDB_G$ tramite $Mongoose_G$ e di restituire il JSON con tutti i dati tramite una callback, in modo da poter generare correttamente la index-page. In caso di fallimento il metodo restituisce un errore tramite una callback di errore.

- `collectionName:String`
  Questo parametro rappresenta il nome della Collection sulla quale effettuare l'estrazione da $MongoDB_G$.
- `callback:function(JSON)`
  Questa callback viene chiamata al termine dell'estrazione dei dati e riceve in input il JSON da restituire alla funzione chiamante.
- `errback:function(MaapError)`
  Questa callback viene chiamata nel caso in cui avvenga un errore nell'estrazione dei dati da $MongoDB_G$. Prende in input l'errore da restituire alla funzione chiamante.

3.7.7 Classe Transformation

Transformation
`+transform(element:Object):Object`

Tabella 27: Classe Transformation

Descrizione

Classe che racchiude tutte le informazioni relative alla modalità con cui i dati prelevati dal database verranno modificati prima di essere inviati al front-end. Tali trasformazioni vengono dichiarate dal developer nel DSL. Questa classe rappresenta una funzione da chiamare sul valore degli attributi

Utilizzo

Questa classe viene creata dalla componente che si occupa di caricare il DSL (interpretandolo o facendone il parsing).

Relazioni con altre classi Assenti

Attributi

Assenti

Metodi

`+transform(element:Object):Object`
Questo metodo prende in input un elemento, applica una trasformazione, e restituisce quest'ultima in output tramite il `return`.

- `element:Object`
  Questo parametro rappresenta l'elemento da che dovrà essere trasformato.

3.7.8 Classe Attribute

Attribute
- label:String - name:String - transformation:function - selectable:Boolean - sortable:Boolean
+Attribute(label:String, name:String, tranformation:function, selectable:Boolean, sortable:Boolean) +getLabel():String +getName():String +getTransformation():function +isSelectable():Boolean +isSortable():Boolean

Tabella 28: Classe Attribute

Descrizione

Classe che racchiude tutte le informazioni relative ad un attributo di una show-page o di una index-page. Tali informazioni vengono dichiarate dal developer nel DSL.

Utilizzo

Questa classe viene creata dalla componente che si occupa di caricare il DSL (interpretandolo o facendone il parsing).

Relazioni con altre classi

Utilizza le classi:

- `Back-end::Lib::Model::DSLModel::Transformation`

Attributi

`- label:String`
Questo campo dati rappresenta la stringa con la quale verrà visualizzata l'intestazione della colonna o della riga nella tabella finale della show-page o della index-page.

`- name:String`
Questo campo dati rappresenta il nome dell'attributo della Collection di riferimento.

`- transformation:function`
Questo campo dati rappresenta una funzione di trasformazione da applicare sull'attributo.

`- selectable:Boolean`
Questo parametro indica se l'attributo è selezionabile, ovvero se nel front-end cliccando su di esso si viene rimandati tramite un link alla show-page del Document riferito.

`- sortable:Boolean`
Questo parametro indica se l'attributo è ordinabile, ovvero se i Document possono essere ordinati secondo esso.

Metodi

`+Attribute(label:String, name:String, tranformation:function, selectable:Boolean, sortable:Boolean)`
Questo metodo è il costruttore della classe. Prima di assegnare i valori viene effettuato un controllo sui parametri passati, in modo che essi siano coerenti con quanto ci si aspetta. Se questo controllo passa allora la classe viene costruita, altrimenti viene sollevata un'eccezione.

- `label:String`
  Questo parametro rappresenta l'etichetta dell'attributo.
- `name:String`
  Questo parametro rappresenta il nome dell'attributo.
- `tranformation:function`
  Questo parametro rappresenta la trasformazione dell'attributo.
- `selectable:Boolean`
  Questo parametro rappresenta il campo selectable della classe.
- `sortable:Boolean`
  Questo parametro rappresenta il campo sortable della classe.

`+getLabel():String`
Questo metodo restituisce il campo *label* della classe.

`+getName():String`
Questo metodo restituisce il campo name della classe.

`+getTransformation():function`
Questo metodo restituisce il campo transformation della classe.

`+isSelectable():Boolean`
Questo metodo restituisce il campo selectable della classe.

`+isSortable():Boolean`
Questo metodo restituisce il campo sortable della classe.

3.8 Componente Back-end::Lib::Utils

3.8.1 Classe Mailer

Mailer
`+Mailer(app:ServerApp)` `+sendEmail(message:Object, callback:function(responseStatus), errback:function(MaapError))`

Tabella 29: Classe Mailer

Descrizione

Classe che si occupa dell'invio di email. È uno dei componenti subsystem class del *Design Pattern*$_G$ *Facade*$_G$ e handler del *Design Pattern*$_G$ *Chain of responsibility*$_G$.

Utilizzo

Viene utilizzata per inviare un'email ad un utente che ha effettuato la richiesta di recupero password.

Relazioni con altre classi Assenti

Attributi

Assenti

Metodi

`+Mailer(app:ServerApp)`
Costruttore che crea il servizio email e rende disponibile l'invio di email tramite `sendEmail()`.

- `app:ServerApp`
  È l'istanza del server che dovrà utilizzare questa classe

`+sendEmail(message:Object, callback:function(responseStatus), errback:function(MaapError))`
Metodo che si occupa di inviare un'email.

- `message:Object`
  Questo oggetto rappresenta il template dell'email da inviare.
- `callback:function(responseStatus)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al termine dell'elaborazione senza errori, dove l'oggetto di tipo responseStatus (tipo appartenente alla libreria `NodeMailer`) contiene informazioni sullo stato di successo.

- `errback:function(MaapError)`
  Questo parametro rappresenta la callback che il metodo dovrà chiamare al verificarsi di un errore.

3.8.2 Classe MaapError

MaapError
- title:String - code:Integer - message:String
+toJson():JSON +toString():String +toError():Error +MaapError(title:String, code:Integer, message:String)

Tabella 30: Classe MaapError

Descrizione

Classe che rappresenta un errore all'interno del package `Back-end::Lib`.

Utilizzo

Viene utilizzata da tutte le classi presente all'interno del package `Back-end::Lib` per rappresentare un errore generato, identificandolo tramite nome, descrizione e codice.

Relazioni con altre classi Assenti

Attributi

`- title:String`
Questo campo dati rappresenta il titolo dell'errore generato in formato stinga.

`- code:Integer`
Campo dato che reppresenta il codice dell'errore.

`- message:String`
Campo dati che rappresenta il messaggio corrispondente all'errore.

Metodi

`+toJson():JSON`
Metodo che ritorna l'errore in formato json.

`+toString():String`
Metodo che effettua una concatenazione dei campi dati dell'errore in formato `String` e la ritorna.

`+toError():Error`
Questo metodo converte l'errore dal tipo `MaapError` al tipo `Error` utilizzato da *Node.js*$_G$ ritornandolo.

`+MaapError(title:String, code:Integer, message:String)`
È il metodo costruttore della classe.

- `title:String`
  Questo parametro rappresenta il titolo del messaggio d'errore.
- `code:Integer`
  Questo parametro rappresenta il codice dell'errore.
- `message:String`
  Questo parametro rappresenta il messaggio dell'errore.

4 Specifica classi del Front-end

4.1 Componente Front-end::Services

4.1.1 Classe UserService

UserService

`+remove()`
`+update(id:Object)`
`+query()`

Tabella 31: Classe UserService

Descrizione

Questa classe permette il recupero della risorsa REST rappresentante l'utente tramite la chiamata /users/{user_id}

Utilizzo

Le funzionalità offerte dalla classe sono:

- elenco dei dati relativi all'utente.
- modifica della password relativa al utente.
- elevare o declassare un utente ad admin
- rimozione dell'utente.

Tali funzionalità richiedono che l'utente sia un admin.

Relazioni con altre classi

Utilizza le classi:

- `Front-end::Model::UserModel`

Attributi

Assenti

Metodi

`+remove()`
Questo metodo si occupa di comunicare al back-end per effettuare l'eliminazione di una risorsa /user.

`+update(id:Object)`
Questo metodo si occupa di comunicare con il back-end per modificare una risorsa passandogli l'id corrispondente.

- `id:Object`
  Questo parametro corrisponde all'id dell'utente di cui modificare la password.

`+query()`
Questo metodo comunica con il back-end per ottenere la risorsa /user.

4.1.2 Classe ProfileService

ProfileService
`+get(id:Object)` `+update(id:Object)`

Tabella 32: Classe ProfileService

Descrizione

Questa classe permette il recupero delle risorsa REST rappresentante il profilo utente tramite la chiamata /profile.

Utilizzo

Le funzionalità offerte dalla classe sono:

- elenco dei dati relativi all'utente (GET);
- modifica dei dati utente (PUT);
- creazione della sessione utente (POST);

– eliminazione della sessione utente (DELETE).

Per la funzionalità di visualizzazione dei dati, di modifica del profilo e di eliminazione della sessione è richiesto che l'utente sia autenticato.

Relazioni con altre classi

Utilizza le classi:

- `Front-end::Model::ProfileModel`

Attributi

Assenti

Metodi

`+get(id:Object)`
Metodo che si occupa di comunicare con il back-end per ottenere il profilo utente.

- `id:Object`
  Parametro che rappresenta l'id dell'utente di cui si richiede i dati profilo.

`+update(id:Object)`
Metodo che comunica con il back-end per richiedere la modifica del profilo utente.

- `id:Object`
  Parametro che rappresenta l'id utente di cui modificare il profilo.

4.1.3 Classe ShowService

ShowService
`+query(collectionName:Object, documentId:Object)`

Tabella 33: Classe ShowService

Descrizione

Questa classe permette il recupero delle risorse REST rappresentanti i Document di una Collection tramite la chiamata /collections/{collection_name}/{document id}

Utilizzo

Le funzionalità offerte dalla classe sono:

– elenco dei dati relativi al Document

– modifica dei dati relativi al Document

– rimozione del Document

Tali funzionalità richiedono che l'utente sia autenticato al sistema.

Relazioni con altre classi Assenti

Attributi

Assenti

Metodi

`+query(collectionName:Object, documentId:Object)`
Metodo che si occupa di richiedere al back-end il contenuto di un documento di una data collection.

- `collectionName:Object`
  Parametro che corrisponde alla collection a cui far riferimento per prelevare il documento.
- `documentId:Object`
  Parametro che indica l'Id del documento da ritornare.

4.1.4 Classe ForgotPasswordService

ForgotPasswordService

Tabella 34: Classe ForgotPasswordService

Descrizione

Questa classe si occupa di inviare al server una richiesta di recupero password tramite la chiamata /password/lost e la conseguente modifica attraverso la chiamata /password/reset.

Utilizzo

La funzionalità offerta dalla classe è quella di interagire col server delegando quest'ultimo all'invio di una mail all'utente per il recupero della password e successivamente alla sua modifica.

Relazioni con altre classi

Utilizza le classi:

- `Front-end::Model::RequestResetModel`

Attributi

Assenti

Metodi

Assenti

4.1.5 Classe IndexService

IndexService
`+query(collectionName:Object)`

Tabella 35: Classe IndexService

Descrizione

Questa classe permette il recupero della risorsa REST rappresentante la Collection tramite la chiamata /collection/{collection_name}

Utilizzo

La funzionalità offerta dalla classe è quella di poter fornire al Controller la lista di Document presenti nella Collection.

Relazioni con altre classi

Utilizza le classi:

- `Front-end::Model::IndexModel`

Attributi

Assenti

Metodi

`+query(collectionName:Object)`
Metodo che comunica col back-end per ottenere la lista di document della collection.

- `collectionName:Object`
  Parametro corrispondente alla collection.

4.1.6 Classe UserListService

UserListService

`+query()`
`+remove(id:Object)`
`+save()`

Tabella 36: Classe UserListService

Descrizione

Questa classe permette il recupero delle risorse REST rappresentanti gli utenti registrati all'applicazione tramite la chiamata /users

Utilizzo

La funzionalità offerta dalla classe è quella di poter fornire al Controller la lista degli utenti presenti nel database delle credenziali. Tale funzionalità richiede che l'utente sia un admin.

Relazioni con altre classi

Utilizza le classi:

- `Front-end::Model::UsersListModel`

Attributi

Assenti

Metodi

`+query()`
Questo metodo si occupa di comunicare col back-end per ottenere la risorsa user che corrisponde alla lista degli utenti nel sistema.

`+remove(id:Object)`
Questo metodo si occupa di comunicare con la componente back-end per eliminare una risorsa /users passando l'id corrispondente.

- `id:Object`
  Parametro che corrisponde all'id utente della risorsa da eliminare.

`+save()`
Metodo che si occupa di comunicare con il back-end per chiedere il salvataggio di una risorsa.

4.1.7 Classe IndexListService

IndexListService
+query()

Tabella 37: Classe IndexListService

Descrizione

Questa classe permette il recupero delle risorse REST rappresentanti le Collections tramite la chiamata / collections.

Utilizzo

La funzionalità offerta dalla classe è quella di poter fornire al Controller la lista delle Collections registrate dallo sviluppatore e presenti nel database delle collections. Tale funzionalità richiede che l'utente sia registrato.

Relazioni con altre classi

Utilizza le classi:

- `Front-end::Model::IndexListModel`

Attributi

Assenti

Metodi

`+query()`
Questo metodo si occupa di comunicare con il back-end per ottenere i nomi delle collection presenti nell'applicazione.

4.2 Componente Front-end::Controllers

4.2.1 Classe LoginController

LoginController
- scope:Object - rootScope:Object - location:Object - ProfileService:Object
+login() +LoginController(rootScope:Object, scope:Object, location:Object, ProfileService:Object)

Tabella 38: Classe LoginController

Descrizione

Classe che gestisce le operazioni e la logica applicativa riguardante la pagina di Login.

Utilizzo

Viene utilizzata per generare la pagina di login all'applicazione. Prima della creazione della view viene effettuato un controllo sull'esistenza di una sessione utente. In caso positivo il controller si occuperà di visualizzare una pagina nella quale l'utente verrà avvertito che un'autenticazione è già stata effettuata, altrimenti si procederà alla pagina di Login predefinita. Una volta che richiede un'autenticazione viene utilizzata classe `Front-End::Services::ProfileService`, la quale si occuperà di comunicare con il Back-End, il quale effettuerà il controllo sulle credenziali e in caso positivo effettuerà l'autenticazione dell'utente.

Relazioni con altre classi

Utilizza le classi:

- `Front-end::View::LoginView`

Attributi

`- scope:Object`
Questo campo dati rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.

`- rootScope:Object`
Questo campo dati rappresenta lo scope radice dell'applicazione. Tutti gli altri scope discendono da questo.

`- location:Object`
Questo campo dati è il servizio che analizza l'URL nella barra degli indirizzi e rende l'URL disponibile all'applicazione. I cambiamenti all'URL nella barra degli indirizzi si riflettono in questo parametro e viceversa.

`- ProfileService:Object`
Questo campo dati rappresenta un riferimento al service utilizzato per l'interazione con il back-end.

Metodi

`+login()`
Questo controller si occupa di comunicare con il ProfileService per richiedere l'autenticazione dell'utente con le credenziali inserite nei campi di testo. Nel caso in cui la richiesta di autenticazione fallisca il metodo si occupa di gestire l'errore tramite messaggio.

`+LoginController(rootScope:Object, scope:Object, location:Object, ProfileService:Object)`
Metodo che rappresenta il costruttore della classe. Si occupa di reperire dalla view l'email e password inserite e di utilizzare il service per richiedere il login dell'utente con le credenziali prelevate.

- `scope:Object`
  Parametro che rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.
- `rootScope:Object`
  Parametro che rappresenta lo scope radice dell'applicazione. Tutti gli altri scope discendono da questo
- `location:Object`
  Parametro il quale rappresenta il servizio che analizza l'URL nella barra degli indirizzi e rende l'URL disponibile all'applicazione. I cambiamenti all'URL nella barra degli indirizzi si riflettono in questo parametro e viceversa.
- `ProfileService:Object`
  Parametro che rappresenta il riferimento al service utilizzato per l'interazione con il back-end.

4.2.2 Classe LogoutController

LogoutController
- scope:Object - rootScope:Object - location:Object - ProfileService:Object
+LogoutController(scope:Object, rootScope:Object, location:Object, ProfileService:Object)

Tabella 39: Classe LogoutController

Descrizione

Classe che gestisce l'operazione di logout di un utente.

Utilizzo

Questa controller si occupa di distruggere la sessione attuale, se esiste, e non genera una view ma reindirizza l'utente automaticamente alla pagina di Login.

Relazioni con altre classi Assenti

Attributi

`- scope:Object`
Questo campo dati rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.

`- rootScope:Object`
Questo campo dati rappresenta lo scope radice, e come padre di tutti gli altri scope permette di accedervi. Il rootScope fornisce le stesse funzionalità degli scope figli.

`- location:Object`
Questo campo dati è il service che analizza l'URL nella barra degli indirizzi e rende l'URL disponibile all'applicazione. I cambiamenti all'URL nella barra degli indirizzi si riflettono in questo parametro e viceversa.

`- ProfileService:Object`
Questo campo dati rappresenta un riferimento al service che si occuperà di comunicare con il back-end ed effettuare il logout dell'utente.

Metodi

`+LogoutController(scope:Object, rootScope:Object, location:Object, ProfileService:Object)`
Metodo costruttore, si occupa di effettuare il logout dell'utente comunicando con il service.

- `scope:Object`
  Questo campo dati rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.
- `rootScope:Object`
  Questo parametro rappresenta lo scope radice, e come padre di tutti gli altri scope permette di accedervi. Il rootScope fornisce le stesse funzionalità degli scope figli.
- `location:Object`
  Questo parametro è il service che analizza l'URL nella barra degli indirizzi e rende l'URL disponibile all'applicazione. I cambiamenti all'URL nella barra degli indirizzi si riflettono in questo parametro e viceversa.
- `ProfileService:Object`
  Parametro rappresentante il riferimento al service che permette l'interfacciamento con il back-end.

4.2.3 Classe ForgotRequestController

ForgotRequestController
- ForgotPasswordService:Object - scope:Object
+ForgotRequestController(scope:Object, ForgotPasswordService:Object)

Tabella 40: Classe ForgotRequestController

Descrizione

Classe che gestisce le operazioni e la logica applicativa riguardante la pagina di richiesta di recupero password.

Utilizzo

Genera una pagina in cui viene visualizzato un campo di testo nel quale l'utente può inserire la propria mail ed effettuare una richiesta di ripristino password. Il controller permette quindi di inviare al Back-end la richiesta attraverso la classe `Front-End::Services::ForgotPasswordService`. Sarà poi compito del *Back-end*$_G$ inviare all'utente una mail contenente il link che bisogna aprire per poter scegliere una nuova password.

Relazioni con altre classi

Utilizza le classi:

- `Front-end::Services::ForgotPasswordService`
- `Front-end::View::ForgotResetView`

Attributi

`- ForgotPasswordService:Object`
Questo campo dati rappresenta il riferimento al service utilizzato per l'interfacciamento al back-end.

`- scope:Object`
Questo campo dati rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.

Metodi

`+ForgotRequestController(scope:Object, ForgotPasswordService:Object)`
Metodo costruttore della classe.

- `scope:Object`
  Questo parametro rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.
- `ForgotPasswordService:Object`
  Questo parametro rappresenta il riferimento al service utilizzato per l'interfaccia: mento al back-end.

4.2.4 Classe ForgotResetController

ForgotResetController
- scope:Object - ForgotPasswordService:Object
+ForgotResetController(scope:Object, ForgotPasswordService:Object)

Tabella 41: Classe ForgotResetController

Descrizione

Classe che gestisce le operazioni e la logica applicativa riguardante la pagina di reset della password.

Utilizzo

Si occupa di generare la pagina di reset, prelevare quindi la nuova password inserita dall'utente nella view e chiamare l'apposito service che si occuperà del reset interagendo con il back-end.

Relazioni con altre classi Assenti

Attributi

`- scope:Object`
Questo campo dati rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.

`- ForgotPasswordService:Object`
Parametro che rappresenta il riferimento al service utilizzato per interagire con il back-end.

Metodi

`+ForgotResetController(scope:Object, ForgotPasswordService:Object)`
Metodo costruttore della classe.

- `scope:Object`
  Questo parametro rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.
- `ForgotPasswordService:Object`
  Parametro che rappresenta il riferimento al service utilizzato per interagire con il back-end.

4.2.5 Classe IndexController

IndexController

`- scope:Object`
`- CollectionService:Object`
`- routeParams:Object`

`+IndexController(scope:Object, rootScope:Object, collectionService:Object)`

Tabella 42: Classe IndexController

Descrizione

Classe che gestisce le operazioni e la logica applicativa riguardante la pagina di gestione della Collection.

Utilizzo

Utilizza la classe `Front-End::Services::IndexService` per popolare correttamente la classe `Front-End::Model::IndexModel`. Quest'ultima fornirà un metodo accessorio attraverso il quale il controller può ottenere i dati e generare la pagina di visualizzazione di tutti i $Document_G$, popolando correttamente lo scope.

Relazioni con altre classi

Utilizza le classi:

- `Front-end::Services::IndexService`
- `Front-end::View::IndexView`

Attributi

`- scope:Object`
Questo campo dati rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.

`- CollectionService:Object`
Campo dati che rappresenta il riferimento al service utilizzato per l'interfacciamento con il back-end.

`- routeParams:Object`
Questo campo dati è il servizio che permette di ottenere informazioni sui correnti parametri di percorso URL.

Metodi

`+IndexController(scope:Object, rootScope:Object, collectionService:Object)`
Metodo costruttore della classe.

- `scope:Object`
  Parametro che rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.
- `rootScope:Object`
  Questo parametro è il servizio che permette di ottenere informazioni sui correnti parametri di percorso URL.
- `collectionService:Object`
  Parametro che rappresenta il riferimento al service utilizzato per l'interfacciamento con il back-end.

4.2.6 Classe UsersListController

UsersListController
- UserListService:Object - UserService:Object - scope:Object - rootScope:Object
+UsersListController(scope:Object, rootScope:Object, UserListService:Object)

Tabella 43: Classe UsersListController

Descrizione

Classe che gestisce le operazioni e la logica applicativa riguardante la pagina di gestione degli utenti.

Utilizzo

Viene utilizzata per generare la pagina di visualizzazione della lista di utenti presenti nel: l'applicazione. In primo luogo utilizzerà la classe `Front-End::Services::UserListService` per popolare la classe `Front-End::Model::UserListModel` dalla quale otterrà in seguito la lista degli utenti attraverso una chiamata a una sua funzione.

Relazioni con altre classi

Utilizza le classi:

- `Front-end::Services::UserService`
- `Front-end::Services::UserListService`
- `Front-end::View::UserListView`

Attributi

`- UserListService:Object`
Questo campo dati rappresenta il service che permette l'interfacciamento con il back-end.

`- UserService:Object`
Questo campo dati rappresenta il service che permette l'interfacciamento con il back-end.

`- scope:Object`
Questo campo dati rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.

`- rootScope:Object`
Questo campo dati rappresenta lo scope radice dell'applicazione. Tutti gli altri scope discendono da questo.

Metodi

`+UsersListController(scope:Object, rootScope:Object, UserListService:Object)`
Metodo costruttore della classe. Si occupa di reperire la lista degli utenti.

- `scope:Object`
  Questo parametro rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.
- `rootScope:Object`
  Questo parametro rappresenta lo scope radice dell'applicazione. Tutti gli altri scope discendono da questo.
- `UserListService:Object`
  Service che permette l'interfacciamento con il back-end.

4.2.7 Classe ShowController

ShowController
- scope:Object - ShowService:Object - routeParams:Object
+ShowController(scope:Object, routeParams:Object, showService:Object)

Tabella 44: Classe ShowController

Descrizione

Classe che gestisce le operazioni e la logica applicativa riguardante la pagina di gestione di un Document.

Utilizzo

Utilizza la classe `Front-End::Services::ShowService` per popolare correttamente la classe `Front-End::Model::ShowModel`, la quale fornirà un metodo accessorio attraverso il quale il controller può ottenere i dati e generare la pagina popolando correttamente lo scope.

Relazioni con altre classi

Utilizza le classi:

- `Front-end::Services::ShowService`
- `Front-end::Model::ShowModel`
- `Front-end::View::ShowView`

Attributi

`- scope:Object`
Questo campo dati rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.

`- ShowService:Object`
Questo campo dati rappresenta il riferimento al service che si occupa di comunicare con il back-end.

`- routeParams:Object`
Questo campo dati rappresenta l'oggetto che permette di recuperare il set di parametri dell'URI corrente.

Metodi

`+ShowController(scope:Object, routeParams:Object, showService:Object)`
Metodo costruttore, si occupa di reperire tramite service le informazioni per poi popolare il model.

- `scope:Object`
  Questo parametro rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.
- `routeParams:Object`
  Parametro che rappresenta l'oggetto che permette di recuperare il set di parametri dell'URI corrente.
- `showService:Object`
  Questo parametro rappresenta il riferimento al service che si occupa di comunicare con il back-end.

4.2.8 Classe DashboardController

DashboardController
- scope:Object - rootScope:Object - IndexListService:Object - location:Object
+DashboardController(scope:Object, rootScope:Object, location:Object, indexListService:Object)

Tabella 45: Classe DashboardController

Descrizione

Classe che gestisce le operazioni e la logica applicativa riguardante la pagina dashboard.

Utilizzo

Viene utilizzata per generare la pagina dashboard, che fungerà da *home* dell'applicazione ovvero la prima pagina che un utente visualizza quando effettua l'autenticazione. Utilizza la classe `Front-End::Services::CollectionListService` per popolare correttamente tutte la classe `Front-End::Model::CollectionListModel`, dalla quale otterrà la lista delle $Collection_G$ registrate nell'applicazione mediante una chiamata a una sua funzione. A questo punto, una volta ottenuti i dati, il controller genera la pagina dashboard, popolando correttamente lo scope con i dati ottenuti.

Relazioni con altre classi

Utilizza le classi:

- `Front-end::Services::IndexListService`

- `Front-end::View::DashboardView`

Attributi

`- scope:Object`
Questo campo dati rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.

`- rootScope:Object`
Questo campo dati rappresenta lo scope radice dell'applicazione. Tutti gli altri scope discendono da questo.

`- IndexListService:Object`
Questo campo dati rappresenta il riferimento al service che permette di comunicare con il back-end.

`- location:Object`
Questo campo dati è il servizio che analizza l'URL nella barra degli indirizzi e rende l'URL disponibile all'applicazione. I cambiamenti all'URL nella barra degli indirizzi si riflettono in questo parametro e viceversa.

Metodi

`+DashboardController(scope:Object, rootScope:Object, location:Object, indexListService:Objec`
Metodo costruttore della classe.

- `scope:Object`
  Parametro che rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.
- `rootScope:Object`
  Questo parametro rappresenta lo scope radice dell'applicazione. Tutti gli altri scope discendono da questo.
- `location:Object`
  Questo parametro rappresenta il servizio che analizza l'URL nella barra degli indirizzi e rende l'URL disponibile all'applicazione. I cambiamenti all'URL nella barra degli indirizzi si riflettono in questo parametro e viceversa.
- `indexListService:Object`
  Questo parametro rappresenta il riferimento al service che permette di comunicare con il back-end.

4.2.9 Classe ProfileController

ProfileController
- scope:Object - ProfileService:Object
+ProfileController(scope:Object, ProfileService:Object)

Tabella 46: Classe ProfileController

Descrizione

Classe che gestisce le operazioni e la logica applicativa riguardante la pagina profilo di un utente.

Utilizzo

Utilizza la classe `Front-End::Services::ProfileService` per popolare la classe `Front-End::Model::ProfileModel` con i dati dell'utente che ha effettuato la richiesta. Quest'ultima classe fornirà un metodo accessorio attraverso il quale il controller potrà prelevare i dati e generare la pagina, popolando correttamente lo scope.

Relazioni con altre classi

Utilizza le classi:

- `Front-end::Services::ProfileService`
- `Front-end::View::ProfileView`

Attributi

`- scope:Object`
Questo campo dati rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.

`- ProfileService:Object`
Questo campo dati rappresenta il riferimento al service utilizzato per comunicare con il back-end per ottenere i dati di cui si necessita.

Metodi

`+ProfileController(scope:Object, ProfileService:Object)`
Metodo costruttore della classe, si occupa di popolare il model tramite il service con i dati utente.

- `scope:Object`
  Questo parametro rappresenta l'oggetto che permette la comunicazione tra la view

ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.

- `ProfileService:Object`
  Questo parametro rappresenta il service utilizzato per comunicare con il back-end.

4.2.10 Classe ProfileEditController

ProfileEditController
- scope:Object - ProfileService:Object
+ProfileEditController(scope:Object, ProfileService:Object)

Tabella 47: Classe ProfileEditController

Descrizione

Classe che gestisce le operazioni di modifica di un utente attraverso la pagina di modifica profilo.

Utilizzo

Utilizza la casse `Front-End::Services::ProfileService` per popolare la classe `Front-End::Model::ProfileModel` con i dati dell'utente. Quest'ultima classe fornirà un metodo accessorio attraverso il quale il controller può ottenere i dati e generare la pagina popolando correttamente lo scope della classe `Front-End::View::ProfileView`.

Relazioni con altre classi

Utilizza le classi:

- `Front-end::View::ProfileEditView`

Attributi

`- scope:Object`
Questo campo dati rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.

`- ProfileService:Object`
Campo dati rappresentante il riferimento al service che il controller utilizza per comunicare con la componente back-end.

Metodi

`+ProfileEditController(scope:Object, ProfileService:Object)`
Metodo costuttore della classe.

- `scope:Object`
  Questo attributo rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.
- `ProfileService:Object`
  Parametro rappresentante il riferimento al service che il controller utilizza per comunicare con la componente back-end.

4.2.11 Classe UserController

UserController
- UserService:Object - scope:Object - rootScope:Object
+deleteUser(user:JSON) +createUser() +UserController(scope:Object, rootScope:Object, userService:Object)

Tabella 48: Classe UserController

Descrizione

Classe che gestisce le operazioni e la logica applicativa riguardante la pagina profilo di un utente visualizzabile dall'admin.

Utilizzo

Utilizza la classe `Front-End::Service::UserService`, che si occupa di popolare la classe `Front-End::Model::UserModel` con i dati dell'utente richiesto. Quest'ultima classe conterrà un metodo accessorio tramite il quale il controller può prelevare i dati e generare la pagina popolando correttamente lo scope della classe `Front-End::View::UserView`.

Relazioni con altre classi

Utilizza le classi:

- `Front-end::View::UserView`

Attributi

`- UserService:Object`
Questo campo dati rappresenta il riferimento al service che permette l'interfacciamento con il back-end.

`- scope:Object`
Questo campo dati rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.

`- rootScope:Object`
Questo campo dati rappresenta lo scope radice dell'applicazione. Tutti gli altri scope discendono da questo

Metodi

`+deleteUser(user:JSON)`
Questo metodo si occupa di comunicare con il service e richiedere l'eliminazione dell'utente indicato.

- `user:JSON`
  Parametro che rappresenta l'utente da eliminare.

`+createUser()`
Questo metodo si occupa di comunicare con il service e richiedere la creazione di un nuovo utente, impostando email, password e livello di autorizzazione.

`+UserController(scope:Object, rootScope:Object, userService:Object)`
Metodo costruttore della classe.

- `scope:Object`
  Parametro che rappresenta l'oggetto che permette la comunicazione tra la view ed il controller, rendendo possibile l'accesso al model mantenendolo sincronizzato, implementando in questo modo il 2-way data binding.
- `rootScope:Object`
  Parametro che rappresenta lo scope radice dell'applicazione. Tutti gli altri scope discendono da questo
- `userService:Object`
  Questo parametro rappresenta il riferimento al service che permette l'interfaccia: mento con il back-end.

4.3 Componente Front-end::Model

4.3.1 Classe ErrorModel

ErrorModel
- type:String - title:String - Content:String

Tabella 49: Classe ErrorModel

Descrizione

È la classe che rappresenta il modello dati dell'errore.

Utilizzo

Utilizzato da tutti i controller per poter accedere alle informazioni riguardanti l'errore.

Relazioni con altre classi Assenti

Attributi

`- type:String`
Definisce se si tratta di un messaggio di conferma o di errore.

`- title:String`
Contiene il titolo del messaggio di alert.

`- Content:String`
Contiene il messaggio d'allerta.

Metodi

Assenti

4.3.2 Classe ProfileModel

ProfileModel
- Utente:JSON

Tabella 50: Classe ProfileModel

Descrizione

È la classe che rappresenta la struttura dati dell'utente.

Utilizzo

Permette al ProfileService di avere una rappresentazione delle informazioni dell'utente da scambiare con il back-end, al ProfileController e al ProfileEditController per ottenere il dati dell'utente da visualizzare nella view della pagina profilo e al ForgotResetController per la modifica della password.

Relazioni con altre classi Assenti

Attributi

`- Utente:JSON`
Contiene i dati dell'utente loggato.

Metodi

Assenti

4.3.3 Classe UserModel

UserModel
`- Utente:JSON`

Tabella 51: Classe UserModel

Descrizione

È la classe che rappresenta la struttura dati dell'utente.

Utilizzo

Fornisce una rappresentazione sotto forma di oggetto delle informazioni scambiate con il back-end e permette allo UserService e allo UserController di poter accedere agli attributi dell'utente.

Relazioni con altre classi Assenti

Attributi

`- Utente:JSON`
Contiene le coppie attributo valore dell'utente selezionato.

Metodi

Assenti

4.3.4 Classe UsersListModel

UsersListModel
- user:JSON

Tabella 52: Classe UsersListModel

Descrizione

È la classe che rappresenta la struttura dati dell'utente.

Utilizzo

Fornisce una rappresentazione sotto forma di oggetto delle informazioni scambiate con il back-end e permette allo UserListService e allo UserListController di poter accedere alla lista degli utenti.

Relazioni con altre classi Assenti

Attributi

`- user:JSON`
Contiene le coppie attributo valore degli utenti.

Metodi

Assenti

4.3.5 Classe RequestResetModel

RequestResetModel
- utente:JSON

Tabella 53: Classe RequestResetModel

Descrizione

È il modello che descrive i dati dell'utente che richiede un recupero della password.

Utilizzo

Fornisce una rappresentazione sotto forma di oggetto delle informazioni scambiate con il back-end e permette al ForgotPasswordService e al ForgotRequestController di poter accedere ai dati dell'utente.

Relazioni con altre classi Assenti

Attributi

- utente:JSON
Contiene i dati dell'utente che richiede un recupero della password.

Metodi

Assenti

4.3.6 Classe IndexModel

IndexModel
- documents:JSON - collectionName:JSON

Tabella 54: Classe IndexModel

Descrizione

È la classe che rappresenta il modello delle Collection.

Utilizzo

Fornisce una rappresentazione sotto forma di oggetto delle informazioni scambiate con il back-end e permette alla CollectionService e alla CollectionController di poter accedere alla lista delle Collections.

Relazioni con altre classi Assenti

Attributi

`- documents:JSON`
Contiene il json contenente i documenti di una data collection.

`- collectionName:JSON`
Contiene il JSON contenente l'elenco delle collection.

Metodi

Assenti

4.3.7 Classe ShowModel

ShowModel
- elements:JSON

Tabella 55: Classe ShowModel

Descrizione

È la classe che rappresenta la struttura dati dei Document relativi ad una Collection.

Utilizzo

Fornisce una rappresentazione sotto forma di oggetto delle informazioni scambiate con il back-end e permette al ShowService e al ShowController di poter accedere agli attributi del Document.

Relazioni con altre classi Assenti

Attributi

`- elements:JSON`
Contiene le coppie attributo-valore del documento richiesto.

Metodi

Assenti

4.3.8 Classe IndexListModel

IndexListModel
- collections:JSON

Tabella 56: Classe IndexListModel

Descrizione

È la classe che rappresenta la struttura dati delle Collections.

Utilizzo

Fornisce una rappresentazione sotto forma di oggetto delle informazioni scambiate con il back-end e permette alla IndexListService e alla DashboardController di poter accedere alla lista delle Collections.

Relazioni con altre classi Assenti

Attributi

- collections:JSON
ontiene l'elenco e la struttura di tutte le collection presenti nel sistema.

Metodi

Assenti

4.4 Componente Front-end::View

4.4.1 Classe IndexView

IndexView
- column[]:Array - val:Array - Id:Array

Tabella 57: Classe IndexView

Descrizione

La classe si occupa di descrivere la pagina che visualizza i documenti della collection selezionata.

Utilizzo

Viene utilizzata dalla classe `IndexController` per generare la index-page di una Collection.

Relazioni con altre classi Assenti

Attributi

- `column[]:Array`
Array contenente gli attributi del documento che lo sviluppatore ha deciso di visualizzare nella index page.

- `val:Array`
Array contenente il valore degli attributi del documento che lo sviluppatore ha deciso di visualizzare nella index page.

- `Id:Array`
Array contenente gli id degli attributi del documento che lo sviluppatore ha deciso di visualizzare nella index page.

Metodi

Assenti

4.4.2 Classe ShowView

ShowView
- rowLabel[]:Array - data[]:Array

Tabella 58: Classe ShowView

Descrizione

Classe descrive la pagina che visualizza le coppie chiave valore del documento selezionato.

Utilizzo

Viene utilizzata dalla classe `ShowController` per generare la show-page di un Document.

Relazioni con altre classi Assenti

Attributi

- `rowLabel[]:Array`
Array contenente le etichette delle chiavi del documento.

- `data[]:Array`
Array contenente i valori delle coppie chiave-valore del documento.

Metodi

Assenti

4.4.3 Classe ForgotRequestView

ForgotRequestView
- email:String

Tabella 59: Classe ForgotRequestView

Descrizione

Classe che rappresenta la pagina che permette all'utente di richiedere il reset della propria password tramite l'inserimento della propria email.

Utilizzo

Relazioni con altre classi Assenti

Attributi

- `email:String`
Campo dati contenente la email relativa all'utente che vuole modificare la propria password.

Metodi

Assenti

4.4.4 Classe DashboardView

DashboardView
- collections[]:Array

Tabella 60: Classe DashboardView

Descrizione

Classe che descrive la pagina che visualizza la dashboard. In questo momento la dasboard contiene la lista delle collection presenti nel sistema.

Utilizzo

Viene utilizzata dalla classe `DashboardController` per generare la pagina dashboard.

Relazioni con altre classi Assenti

Attributi

`- collections[]:Array`
Array contenente i nomi delle collection presenti.

Metodi

Assenti

4.4.5 Classe UserListView

UserListView
- user:JSON

Tabella 61: Classe UserListView

Descrizione

Questa classe si occupa di rappresentare la pagina contenente l'elenco di tutti gli utenti presenti nel sistema.

Utilizzo

Viene utilizzata dalla classe `Front-End::Controller::UserListController` per generare la pagina di visualizzazione degli utenti.

Relazioni con altre classi Assenti

Attributi

`- user:JSON`
Questo campo dati rappresenta la lista di utenti registrati all'applicazione.

Metodi

Assenti

4.4.6 Classe UserView

UserView
- level:Integer - role:String - email:String

Tabella 62: Classe UserView

Descrizione

Questa classe si occupa di descrivere la pagina di visualizzazione delle informazioni sull'utente selezionato.

Utilizzo

Viene utilizzata dalla classe `Front-End::UserController` per generare correttamente la pagina di visualizzazione delle informazione di un utente.

Relazioni con altre classi Assenti

Attributi

`- level:Integer`
Questo campo dati rappresenta il livello di permesso dell'utente in forma numerica.

`- role:String`
Questo campo dati rappresenta il livello di permesso dell'utente in forma testuale.

`- email:String`
Questo metodo rappresenta l'indirizzo email dell'utente.

Metodi

Assenti

4.4.7 Classe LoginView

LoginView
- email:String - password:String

Tabella 63: Classe LoginView

Descrizione

Questa classe si occupa di descrivere la pagina di login dell'applicazione mettendo a disposizione dell'utente un form all'interno del quale inserire email e password. Viene inoltre messo a disposizione un link per richiedere il ripristino della password.

Utilizzo

Viene utilizzata dalla classe `LoginController` per generare la pagina di Login dell'applicazione.

Relazioni con altre classi Assenti

Attributi

`- email:String`
Questo parametro rappresenta l'email inserita dall'utente nel campo email del form della pagina di Login.

`- password:String`
Questo parametro rappresenta la password inserita dall'utente nel campo password del form della pagina di Login.

Metodi

Assenti

4.4.8 Classe ProfileView

ProfileView
- email:String - id:String + password:String

Tabella 64: Classe ProfileView

Descrizione

Classe che rappresenta la pagina che visualizza le informazioni dell'utente attualmente autenticato.

Utilizzo

Viene utilizzata dalla classe `Front-end::Controller::ProfileController` per generare la pagina di visualizzazione del profilo dell'utente generato.

Relazioni con altre classi Assenti

Attributi

`- email:String`
Campo dati che contiene l'email dell'utente attualmente loggato.

`- id:String`
Campo dati che contiene l'id dell'utente attualmente loggato.

`+ password:String`
Questo campo dati rappresenta la password dell'utente.

Metodi

Assenti

4.4.9 Classe ForgotResetView

ForgotResetView
- password:String

Tabella 65: Classe ForgotResetView

Descrizione

Classe che rappresenta la pagina che permette all'utente di resettare la propria password. Viene reindirizzato a questa pagine tramite un link presente nell'email ricevuta a seguito della compilazione di ForgotRequestView.

Utilizzo

Viene utilizzato dalla classe `Front-End::Controller::ForgotResetController` per generare correttamente la pagina di reset della password.

Relazioni con altre classi Assenti

Attributi

`- password:String`
Questo campo dati rappresente la nuova password.

Metodi

Assenti

4.4.10 Classe ProfileEditView

ProfileEditView
- email:String - id:String + password:String

Tabella 66: Classe ProfileEditView

Descrizione

Questa classe descrive la pagina che si occupa di modificare i dati dell'utente attualmente autenticato.

Utilizzo

Viene utilizzato dalla classe `Front-end::Controller::ProfileEditController` per generare correttamente la pagina di modifica profilo.

Relazioni con altre classi Assenti

Attributi

`- email:String`
Campo dati che contiene l'email dell'utente attualmente loggato.

`- id:String`
Campo dati che contiene l'id dell'utente attualmente loggato.

`+ password:String`
Questo campo dati rappresenta la password che dovrà essere modificata dall'utente.

Metodi

Assenti

5 Tracciamento

5.1 Tracciamento requisiti-classi

Requisito	Classe
RA1O 1	Front-end::Controllers::LoginController
RA1O 1.1	Front-end::Controllers::LoginController
RA1O 1.2	Front-end::Controllers::LoginController
RA1O 1.3	Back-end::Lib::Controller::Middleware::Authentication Back-end::Lib::Controller::Service::UserService Back-end::Lib::Model::UserModel
RA1O 1.3.1	Front-end::Model::ErrorModel
RA1O 1.3.2	Back-end::Lib::Controller::Middleware::Authentication Back-end::Lib::Controller::Service::ProfileService Front-end::Controllers::DashboardController
RA1O 2	Back-end::Lib::Controller::Service::ForgotService Front-end::Services::ForgotPasswordService Front-end::Model::RequestResetModel
RA1O 2.1	Front-end::Controllers::ForgotRequestController
RA1O 2.2	Back-end::Lib::Controller::Service::ForgotService Back-end::Lib::Utils::Mailer Back-end::Lib::View::ForgotMailView
RA1D 3	Front-end::Controllers::DashboardController
RA1O 4	Front-end::Controllers::IndexController

RA1O 4.1	Back-end::Lib::Controller::Service::IndexService Front-end::Controllers::IndexController Front-end::Model::IndexListModel
RA1O 4.1.1	Front-end::Controllers::IndexController
RA1O 5	Front-end::Controllers::ShowController Front-end::Model::ShowModel
RA1O 5.3	Back-end::Lib::Controller::Service::ShowService Front-end::Controllers::ShowController
RA1O 6	Back-end::Lib::Controller::Service::UserService Front-end::Controllers::UsersListController
RA1O 6.1	Front-end::Controllers::UserController Back-end::Lib::Controller::Service::UserService Back-end::Lib::Model::UserModel
RA1O 6.1.1	Front-end::Controllers::UserController
RA1O 6.1.1.1	Front-end::Controllers::UserController
RA1O 6.1.1.2	Front-end::Controllers::UserController
RA1O 6.1.1.3	Front-end::Controllers::UserController
RA1O 6.1.2	Back-end::Lib::Controller::Service::UserService Back-end::Lib::Model::UserModel Front-end::Controllers::UserController Front-end::Model::UserModel
RA1O 6.2	Front-end::Controllers::UsersListController
RA1O 6.2.1	Back-end::Lib::Controller::Service::UserService Front-end::Controllers::UserController
RA1O 6.2.2	Front-end::Controllers::UserController

RF1O 7	Back-end::Lib::Model::DSLModel::DSLDomain Back-end::Lib::Model::DSLModel::DSLInterpreterStrategy Back-end::Lib::Model::DSLModel::DSLInterpreterStrategy:: ConcreteDSLInterpreter
RF1O 8	Back-end::Lib::ServerApp
RF1O 8.1	Back-end::DeveloperProject::ProjectApp
RF1O 8.1.1	Back-end::DeveloperProject::ProjectApp
RF1O 8.1.2	Back-end::DeveloperProject::ProjectApp Back-end::DeveloperProject::Config::ProjectConfig
RF1O 9	Back-end::Lib::Model::DSLModel::DSLCollectionModel
RF1O 9.1	Back-end::Lib::Model::DSLModel::IndexModel
RF1O 9.1.1	Back-end::Lib::Model::DSLModel::Attribute
RF1O 9.1.2	Back-end::Lib::Model::DSLModel::IndexModel
RF1O 9.1.3	Back-end::Lib::Model::DSLModel::IndexModel
RF1O 9.1.4	Back-end::Lib::Model::DSLModel::IndexModel
RF1O 9.1.5	Back-end::Lib::Model::DSLModel::IndexModel
RF1O 9.1.6	Back-end::Lib::Model::DSLModel::IndexModel
RF1O 9.1.7	Back-end::Lib::Model::DSLModel::Trasform
RF1O 9.2	Back-end::Lib::Model::DSLModel::ShowModel
RF1O 9.2.1	Back-end::Lib::Model::DSLModel::Attribute

RF1O 9.2.2	Back-end::Lib::Model::DSLModel::Attribute
RF1O 9.2.3	Back-end::Lib::Model::DSLModel::ShowModel
RF1O 9.2.4	Back-end::Lib::Model::DSLModel::Trasform
RF1O 14	Back-end::DeveloperProject::Config::ProjectConfig
RF1O 14.1	Back-end::DeveloperProject::Config::ProjectConfig
RA1O 18	Back-end::Lib::Model::DSLModel::DSLInterpreterStrategy:: ConcreteDSLInterpreter

Tabella 67: Requisiti-Classi

5.2 Tracciamento metodi-test

Classe e Metodo	Test
Back-end::Lib::Controller::Middleware::Router::handler()	
Back-end::Lib::Controller::Middleware::Router::init()	TU - 67
Back-end::Lib::Controller::Middleware::Authentication::handler()	
Back-end::Lib::Controller::Middleware::Authentication::authenticate()	
Back-end::Lib::Controller::Middleware::Authentication::init()	
Back-end::Lib::Controller::Middleware::Authentication::requireLogged()	TU - 80
Back-end::Lib::Controller::Service::ProfileService::login()	TU - 72
Back-end::Lib::Controller::Middleware::Authentication::requireAdmin()	TU - 79
Back-end::Lib::Controller::Service::UserService::registerUser()	TU - 49
Back-end::Lib::Controller::Service::UserService::insertUser()	TU - 50
Back-end::Lib::Controller::Service::UserService::userIdShowPage()	TU - 51
Back-end::Lib::Controller::Service::UserService::updateLevel()	TU - 53
Back-end::Lib::Utils::Mailer::Mailer()	TU - 54
Back-end::Lib::Utils::Mailer::sendEmail()	
Back-end::Lib::Controller::Middleware::Authentication::requireNotLogged()	TU - 81
Back-end::Lib::Controller::Middleware::Authentication::requireSuperAdmin()	TU - 82
Back-end::Lib::Controller::Middleware::MiddlewareLoader::init()	
Back-end::Lib::Controller::Service::ForgotService::passwordResetRequest()	TU - 77
Back-end::Lib::Model::DSLModel::DSLCollectionModel:: DSLCollectionModel()	TU - 28
Back-end::Lib::Model::DSLModel::DSLCollectionModel::getCollectionName()	TU - 29
Back-end::Lib::Model::DSLModel::DSLCollectionModel::getIndexModel()	TU - 30
Back-end::Lib::Model::DSLModel::DSLCollectionModel::getShowModel()	TU - 31
Back-end::Lib::Controller::Service::ServiceFactory::getCollectionController()	TU - 56
Back-end::Lib::Controller::Service::ServiceFactory::getProfileController()	TU - 57
Back-end::Lib::Controller::Service::ServiceFactory::getAuthController()	TU - 58
Back-end::Lib::Controller::Service::ServiceFactory::getForgotController()	TU - 59
Back-end::Lib::Controller::Service::ServiceFactory::getUserController()	TU - 60
Back-end::Lib::Controller::Service::ServiceFactory::getShowController()	TU - 61

Back-end::Lib::Controller::Service::ServiceFactory::getIndexController()	TU - 62
Back-end::Lib::View::ForgotMailView::buildForgotMail()	TU - 63
Back-end::Lib::Controller::Middleware::ErrorHandler::handler()	TU - 68
Back-end::Lib::Controller::Middleware::DSLLoaderHandler:: browseFileSystem()	TU - 78
Back-end::Lib::Controller::Service::ProfileService::logout()	TU - 73
Back-end::Lib::Controller::Middleware::NotFoundHandler::handler()	TU - 76
Back-end::Lib::Controller::Middleware::DSLLoaderHandler::init()	TU - 66
Back-end::Lib::Model::DSLModel::DSLDomain::getErrors()	TU - 16
Back-end::Lib::Utils::MaapError::toString()	TU - 7
Back-end::Lib::Model::UserModel::getUserById()	TU - 24
Back-end::Lib::Config::getServerPort()	
Back-end::Lib::Config::getServerStaticPath()	
Back-end::Lib::Config::getUserDbUri()	
Back-end::Lib::ServerApp::start()	
Back-end::Lib::Model::UserModel::getUserList()	TU - 18
Back-end::Lib::Utils::MaapError::toJson()	TU - 6
Back-end::DeveloperProject::ProjectApp::start()	
Back-end::Lib::Config::getEnvironment()	
Back-end::Lib::Config::getDataDbUri()	
Back-end::Lib::Config::getSmtpService()	
Back-end::Lib::Config::getSmtpAuth()	
Back-end::Lib::Model::DSLModel::DSLDomain::registerCollection()	TU - 14
Back-end::Lib::Model::DSLModel::DSLDomain::getCollectionModel()	TU - 15
Back-end::Lib::Model::UserModel::registerUser()	TU - 19
Back-end::Lib::Model::UserModel::deleteUser()	TU - 22
Back-end::Lib::Model::UserModel::updateLevel()	TU - 20
Back-end::Lib::Model::UserModel::createUser()	TU - 21
Back-end::Lib::Model::UserModel::updatePassword()	TU - 23
Back-end::Lib::Model::DSLModel::DSLInterpreterStrategy:: ConcreteDSLInterpreter::init()	TU - 26

Back-end::Lib::Model::DSLModel::DSLInterpreterStrategy:: ConcreteDSLInterpreter::loadDSLFile()	TU - 27
Back-end::Lib::Utils::MaapError::toError()	TU - 8
Back-end::Lib::Controller::Service::UserService::usersList()	TU - 52
Back-end::Lib::Controller::Service::ShowService::getShowPage()	TU - 67
Back-end::Lib::Controller::Service::ShowService::deleteDocument()	TU - 71
Back-end::Lib::Controller::Service::ProfileService::getProfile()	TU - 74
Back-end::Lib::Controller::Service::ProfileService::updatePassword()	TU - 75
Back-end::Lib::Model::DSLModel::Attribute::getLabel()	TU - 43
Back-end::Lib::Model::DSLModel::Attribute::getName()	TU - 44
Back-end::Lib::Model::DSLModel::ShowModel::ShowModel()	TU - 38
Back-end::Lib::Model::DSLModel::Attribute::getTransformation()	TU - 45
Back-end::Lib::Model::DSLModel::Attribute::isSelectable()	TU - 46
Back-end::Lib::Utils::MaapError::MaapError()	TU - 5
Front-end::Services::IndexListService::query()	TU - 87
Front-end::Services::UserListService::remove()	TU - 89
Back-end::Lib::Model::DSLModel::DSLDomain::DSLDomain()	TU - 12
Back-end::Lib::Model::DSLModel::DSLDomain::loadDSLFile()	TU - 13
Back-end::Lib::Model::UserModel::init()	TU - 17
Back-end::Lib::Model::DSLModel::DSLInterpreterStrategy:: ConcreteDSLInterpreter::DSLConcreteStrategy()	TU - 25
Back-end::Lib::Model::DSLModel::DSLCollectionModel::setIndexModel()	TU - 32
Back-end::Lib::Model::DSLModel::DSLCollectionModel::setShowModel()	TU - 33
Back-end::Lib::Model::DSLModel::IndexModel::IndexModel()	TU - 34
Back-end::Lib::Model::DSLModel::IndexModel::addAttribute()	TU - 35
Back-end::Lib::Model::DSLModel::IndexModel::getAttributes()	TU - 36
Back-end::Lib::Model::DSLModel::IndexModel::getData()	TU - 37
Back-end::Lib::Model::DSLModel::ShowModel::addAttribute()	TU - 39
Back-end::Lib::Model::DSLModel::ShowModel::getAttributes()	TU - 40
Back-end::Lib::Model::DSLModel::ShowModel::getData()	TU - 41
Back-end::Lib::Config::getDSLPath()	
Back-end::Lib::Model::DSLModel::Attribute::Attribute()	TU - 42

Back-end::Lib::Model::DSLModel::Attribute::isSortable()	TU - 47
Back-end::Lib::Controller::Service::UserService::deleteUser()	TU - 48
Back-end::Lib::Controller::Service::IndexService::getIndexPage()	TU - 54
Back-end::Lib::Controller::Middleware::DSLLoaderHandler:: DSLLoaderHandler()	TU - 63
Back-end::Lib::ServerApp::ServerApp()	TU - 4
Back-end::Lib::Controller::Service::CollectionListService::list()	
Back-end::Lib::Controller::Service::CollectionListService::getDashboard()	TU - 107
Front-end::Services::UserListService::query()	TU - 90
Front-end::Controllers::LoginController::LoginController()	TU - 94
Front-end::Services::UserService::query()	TU - 82
Front-end::Services::UserService::update()	TU - 83
Front-end::Services::UserService::remove()	TU - 84
Front-end::Services::ShowService::query()	TU - 85
Front-end::Services::IndexService::query()	TU - 86
Front-end::Services::UserListService::save()	TU - 88
Front-end::Services::ProfileService::update()	TU - 91
Front-end::Services::ProfileService::get()	TU - 92
Front-end::Controllers::LoginController::login()	TU - 93
Front-end::Controllers::LogoutController::LogoutController()	TU - 95
Front-end::Controllers::ShowController::ShowController()	TU - 98
Front-end::Controllers::ForgotResetController::ForgotResetController()	TU - 105
Front-end::Controllers::ProfileEditController::ProfileEditController()	TU - 106
Front-end::Controllers::UserController::createUser()	TU - 10
Front-end::Controllers::UserController::deleteUser()	TU - 96
Front-end::Controllers::UserController::UserController()	TU - 97
Front-end::Controllers::ProfileController::ProfileController()	TU - 104
Front-end::Controllers::UsersListController::UsersListController()	TU - 103
Front-end::Controllers::ForgotRequestController::ForgotRequestController()	TU - 101
Front-end::Controllers::IndexController::IndexController()	TU - 100
Front-end::Controllers::DashboardController::DashboardController()	TU - 99

Back-end::Lib::Model::DSLModel::DSLInterpreterStrategy::*loadDSLFile()*()	
Back-end::Lib::Model::DSLModel::Transformation::transform()	

Tabella 68: Metodi-Test