MaaS - MongoDB as an admin Service

matrioska.io.go@gmail.com

Definizione di Prodotto V2.0.0

Nome del documento	Definizione di Prodotto
Versione del Documento	2.0.0
Data Creazione	13/07/2016
Redazione	Santi Guido
Verifica	D'Amico Roberto
Approvazione	Berselli Marco
Uso	*Esterno*
Destinatari	*RedBabel*, Prof. Tullio Vardanega, Prof. Riccardo Cardin.

Registro delle modifiche

Versione	Autore	Ruolo	Data	Descrizione
2.0.0	Berselli Marco	Responsabile	09/08/2016	Approvazione
1.3.2	D'Amico Roberto	Verificatore	08/08/2016	Verifica Finale
1.3.1	Zamberlan Sebastiano	Progettista	06/09/2016	Correzioni errori riscontrati in sede di verifica
1.3.0	D'Amico Roberto	Verificatore	05/09/2016	Verifica Sezioni 4.5 e 4.6
1.2.3	Zamberlan Sebastiano	Progettista	05/09/2016	Aggiornamento Utils sezione 4.6
1.2.2	Zamberlan Sebastiano	Progettista	04/09/2016	Aggiornamento Model sezione 4.5
1.2.1	Santi Guido	Progettista	02/09/2016	Correzioni errori riscontrati in sede di verifica
1.2.0	D'Amico Roberto	Verificatore	03/09/2016	Verifica Sezioni 4.3 e 4.4
1.1.3	Santi Guido	Progettista	03/09/2016	Aggiornamento Containers sezione 4.4
1.1.2	Santi Guido	Progettista	03/09/2016	Aggiornamento Components sezione 4.3
1.1.1	Santi Guido	Progettista	02/09/2016	Correzioni errori riscontrati in sede di verifica
1.1.0	D'Amico Roberto	Verificatore	03/09/2016	Verifica Sezioni 4.1 e 4.2
1.0.2	Santi Guido	Progettista	02/09/2016	Aggiornamento Reducers sezione 4.2
1.0.1	Santi Guido	Responsabile	02/09/2016	Aggiornamento Actions sezione 4.1
1.0.0	Santi Guido	Responsabile	15/08/2016	Approvazione
0.5.0	Berselli Marco	Verificatore	15/08/2016	Verifica finale
0.4.1	D'Amico Roberto	Analista	12/08/2016	Inclusione della classe MethodDisabling in Back-End
0.4.0	D'Amico Roberto	Analista	11/08/2016	Stesura Back-End

Versione	Autore	Ruolo	Data	Descrizione
0.3.0	D'Amico Roberto	Analista	29/07/2016	Stesura Front-End
0.2.0	Santi Guido	Amministratore	14/07/2016	Stesura introduzione
0.1.0	Santi Guido	Amministratore	13/07/2016	Stesura iniziale

Indice

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Questo documento ha il compito di descrivere la progettazione in dettaglio definita per il progetto MaaS. Le informazioni descritte in questo documento sono basate su quanto descritto nella *"Specifica Tecnica v2.0.0"*. Il gruppo si servirà di quanto affermato in questo documento per l'attività di codifica.

1.2 Scopo del prodotto

L'obiettivo del prodotto *software*$_g$ è adattare il *framework*$_g$ *MongoDB*$_g$ as an admin Platform (*MaaP*$_g$), sviluppato dal gruppo SteakHolders durante l'attività accademica di *progetto*$_g$ nel corso di Ingegneria del Software dell'a.a. 2013/2014, per offrire il prodotto come *servizio web*$_g$.

1.3 Glossario

Con il presente viene consegnato anche un *"Glossario v3.0.0"* con lo scopo di facilitare la lettura dei documenti formali. Ogni acronimo o termine tecnico accompagnato con una g a pedice sarà quindi integrato con ulteriori informazioni da ricercarsi nel suddetto Glossario.

1.4 Riferimenti

1.4.1 Normativi

- Norme di *Progetto*$_g$: *"Norme di Progetto v3.0.0"*;
- Specifica Tecnica: *"Specifica Tecnica v2.0.0"*;
- *Capitolato*$_g$ d'appalto C4: MaaS "MongoDB as an admin Service". Reperibile all'*indirizzo web*$_g$: http://www.math.unipd.it/~tullio/IS-1/2015/Progetto/C4.pdf.

1.4.2 Informativi

- Redux: http://redux.js.org;
- ECMAScript6: http://es6-features.org;
- Loopback.io: https://loopback.io;
- Node.js: https://nodejs.org;
- MongoDB: https://docs.mongodb.com;
- SuperAgent: https://visionmedia.github.io/superagent;
- Sweet.js: http://sweetjs.org;

2 Standard di progetto

2.1 Standard di progettazione architetturale

Gli standard di progettazione sono definiti internamente alla *"Specifica Tecnica v2.0.0"*.

2.2 Standard di documentazione del codice

Gli standard da seguire per la stesura della documentazione del codice sono definiti internamente alle *"Norme di Progetto v3.0.0"*.

2.3 Standard di denominazione di entità e relazioni

Le norme tipografiche relative ai nomi delle entità sono definite nelle *"Norme di Progetto v3.0.0"*.

2.4 Standard di programmazione

Gli standard di programmazione sono definiti e descritti nelle *"Norme di Progetto v3.0.0"*.

2.5 Strumenti di lavoro

Gli strumenti di lavoro da utilizzare durante la codifica sono descritti all'interno del documento *"Norme di Progetto v3.0.0"*.

3 Specifica classi del Back-end

3.1 Tokens

Nella sezione successiva vengono citati più volte i cosiddetti tokens. Ecco una loro definizione più dettagliata:

Token di autenticazione
Il più complesso e completo dei token, creato utilizzando una codifica JWT. Viene salvato e riconosciuto da Loopback. Le informazioni conservate al suo interno sono:

- *email*: Indirizzo email, nonché codice univoco dell'utente;
- *company*: Nome della compagnia, nonché codice univoco dell'azienda a cui l'utente appartiene;
- *duty*: Ruolo dell'utente all'interno dell'azienda;
- *createdAt*: Momento della creazione del token, in millisecondi. Necessario per non ottenere token sempre uguali e facilmente decodificabili.

Token password
Token JWT molto semplice il cui breve TTL non viene gestito da Loopback ma da dalla classe Account.

- *email*: Indirizzo email, nonché codice univoco dell'utente;
- *createdAt*: Momento della creazione del token, in millisecondi. Necessario sia per sicurezza che per controllarne il TTL, non gestito da Loopback.

Token di verificazione
A differenza dagli altri token questo non viene generato utilizzando una tecnologia JWT. Si tratta di una stringa randomica che viene memorizzata come attributo temporaneo di un Account, fino a verifica effettuata.

3.2 Hooks

Alcune delle classi del BackEnd sono composte da funzioni chiamate Hooks. In poche parole sono operazioni fatte prima o dopo un'altra determinata funzione, al fine di verificare accessi e formattare meglio input e output. Tutti gli hooks sono da ritenersi metodi privati con la seguente struttura:

- *-AfterUserCreate(context:Object,response:Object,next:function)*: [...]

Al fine di migliorare la leggibilità del documento, gli hooks sono posti in un elenco separato e ne viene riportato solo il nome e la descrizione.

3.3 Models

3.3.1 BackEnd :: Common :: Models :: Account

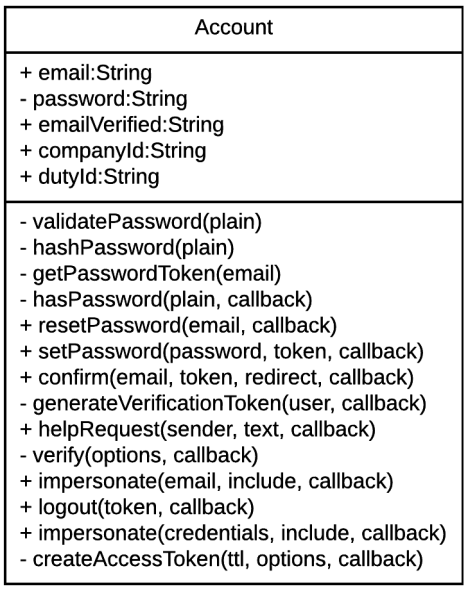

Fig 1: Account

Descrizione

Classe che raccoglie tutti i metodi remoti chiamabili tramite API sugli account utente.

Utilizzo

Questa classe estende dalla classe base loopback PersistedModel, estendere la classe User comportava una serie di limitazioni esagerata.

Attributi

- *+email:string*: Indirizzo email nonché codice univoco.
- *-password:string*: Parola segreta per autenticarsi nel sistema.
- *+emailVerified:boolean*: Valore booleano che indica se l'utente ha accettato l'invito.
- *+subscribedAt:Date*: Data e ora dell'invio dell'invito.
- *+companyId:string*: Codice univoco dell'azienda al quale l'utente appartiene.
- *+dutyId:string*: Codice univoco del ruolo che l'utente ricopre all'interno dell'azienda.

Metodi

- *-validatePassword(plain:string)*: Controlla che la password rispetti gli standard definiti;

- *-hashPassword(plain:string)*: Cripta una password in chiaro in una stringa codificata da 128 caratteri;
- *-getPasswordToken(email:string)*: Genera un token password personale;
- *-hasPassword(plain:string,cb:function)*: Verifica attraverso una libreria crittografica che una password in chiaro corrisponda a quella dell'utente;
- *+resetPassword(email:string,cb:function)*: Spedisce una email contenente un token password integrato in un link ad un form di reimpostazione della password;
- *+setPassword(pass:string,token:string,cb:function)*: Imposta la password di un utente ad un nuovo valore a fronte del corrispondente token password;
- *+confirm(uid:string,token:string,redirect:string,cb:function)*: Rende valido un account invitato a fronte del corrispondente token di verificazione;
- *-generateVerificationToken(user:Object,cb:function)*: Genera un token di verificazione per un utente invitato;
- *+helpRequest(sender:string,text:text,cb:function)*: Spedisce a MaaS una email contente il testo e la email inserite da un utente;
- *-verify(option:Object,cb:function)*: Spedisce una email di invito ad un dato account, corredata di link per la verificazione;
- *+impersonate(email:string,include:Object,cb:function)*: Fornisce un token di autenticazione di un qualsiasi utente, senza fornire la password;
- *+logout(tokenId:string,cb:function)*: Distrugge il token di validazione utilizzato, rendendolo inutilizzabile in futuro;
- *+login(credentials:Object,include:Object,cb:function)*: Fornisce un token di autenticazione personale a fronte di credenziali corrette;
- *-createAccessToken(ttl:number,option:Object,cb:function)*: Genera un token di autenticazione personale.

Hooks

- *BeforeCreate*: Viene impostato l'attributo *emailVerified* a falso, per motivi di sicurezza;
- *BeforeSetPassword*: Viene verificato che il token password utilizzato non sia scaduto;
- *BeforeImpersonate*: Viene verificato che il token di autenticazione appartenga ad un SuperAdmin.

Relazioni con altre classi

Le relazioni seguenti sono di fatto riferimenti fra modelli di dati
BackEnd :: Common :: Models :: Company - l'azienda di appartenenza
BackEnd :: Common :: Models :: Duty - il ruolo dell'utente

3.3.2 BackEnd :: Common :: Models :: Company

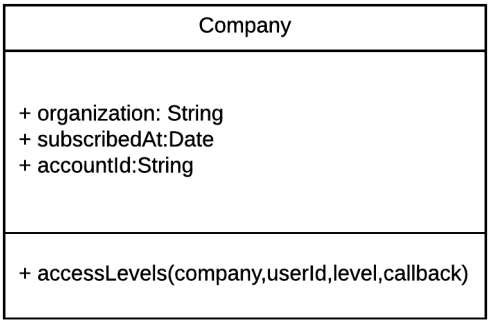

Fig 2: Company

Descrizione

Classe che raccoglie tutti i metodi remoti chiamabili tramite API sulle aziende iscritte a MaaS.

Utilizzo

Questa classe estende dalla classe base loopback PersistedModel.

Attributi

- *+organization:string*: Nome della compagnia nonché codice univoco.
- *+subscribedAt:Date*: Data e ora della creazione dell'istanza aziendale.
- *+accountId:string*: Indirizzo email dell'account del proprietario.

Metodi

- *+accessLevels(company:string,userId:string,level:number,callback:function*: Aggiorna ad nuovo valore l'attributo dutyId di un utente.

Hooks

- *AfterUserCreate*: Successivamente all'invito di un utente questo metodo spedisce una mail di invito.
- *BeforeDslsCreate*: Creare DSLI private è permesso a tutti gli utenti autenticati. Creare DSLI pubbliche è permesso solo agli amministratori della stessa azienda;
- *BeforeDslsGet*: Ottenere i dati di tutte le DSLI al quale si ha accesso è permesso a tutti gli utenti autenticati;
- *BeforeDslsUpdateByID*: Cambiare i dati di una DSLI, ad eccezione di permessi e autore, è permesso a chiunque disponga del permesso di modifica. Gli amministratori possono inoltre modificare i permessi;
- *BeforeDslsDestroyByID*: Eliminare una DSLI è permesso a chiunque disponga del permesso di modifica;

- *BeforeDsIsFindByID*: Ottenere i dati di una DSLI, tra cui il codice, è permesso a chiunque disponga del permesso di esecuzione;
- *BeforeDatabaseCreate*: Collegarsi a nuovi database è permesso solo agli amministratori della stessa azienda;
- *BeforeDatabaseGet*: Ottenere i dati dei database di un'azienda è permesso a tutti gli utenti della stessa azienda;
- *BeforeDatabaseUpdateByID*: Cambiare i dati di un database è permesso solo agli amministratori della stessa azienda;
- *BeforeDatabaseDestroyByID*: Eliminare i dati di un database è permesso solo agli amministratori della stessa azienda;
- *BeforeDatabaseFindByID*: Ottenere i dati di un database è permesso a tutti gli utenti della stessa azienda;
- *BeforeUserCreate*: Invitare gli utenti è permesso solo agli amministratori della stessa azienda;
- *BeforeUserGet*: Ottenere i dati degli utenti di un'azienda è permesso solo agli amministratori della stessa azienda;
- *BeforeAccessLevels*: Cambiare i livelli di accesso di un account di un utente membro o amministratore è permesso solo agli amministratori della stessa azienda;
- *BeforeUserDestroyByID*: Eliminare l'account di un utente membro o amministratore è permesso solo agli amministratori della stessa azienda;
- *BeforeUserFindByID*: Ottenere i dati di un utente è permesso solo agli amministratori della stessa azienda.

Relazioni con altre classi

Le relazioni seguenti sono di fatto riferimenti fra modelli di dati
BackEnd :: Common :: Models :: Account - l'utente proprietario

3.3.3 BackEnd :: Common :: Models :: Database

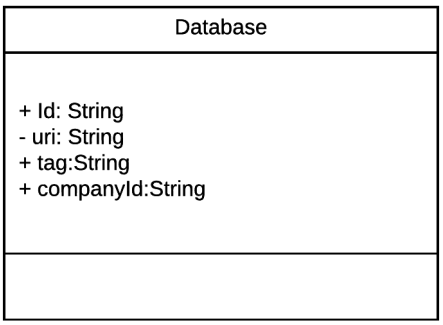

Fig 3: Database

Descrizione

Classe che raccoglie tutti i metodi remoti chiamabili tramite API sui database aziendali.

Utilizzo

Questa classe estende dalla classe base loopback PersistedModel.

Attributi

- *+Id:string*: Codice univoco generato automaticamente.
- *-uri:string*: Stringa di connessione includente dati sensibili.
- *+tag:string*: Nome mnemonico per distinguere i database.
- *+companyId:string*: Codice univoco dell'azienda al quale il database appartiene.

Relazioni con altre classi

Le relazioni seguenti sono di fatto riferimenti fra modelli di dati
BackEnd :: Common :: Models :: Company - l'azienda di appartenenza

3.3.4 BackEnd :: Common :: Models :: DSLI

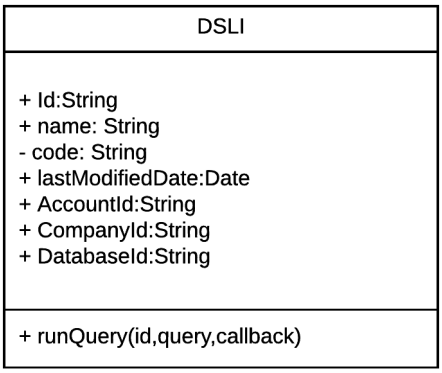

Fig 4: DSLI

Descrizione

Classe che raccoglie tutti i metodi remoti chiamabili tramite API su una DSLI.

Utilizzo

Questa classe estende dalla classe base loopback PersistedModel.

Attributi

- *+Id:string*: Codice univoco generato automaticamente.
- *+name:string*: Nome mnemonico per distinguerla dalle altre DSLI.
- *-code:string*: Codice scritto in DSL.
- *+lastModifiedDate:Date*: Data di ultima modifica.
- *+accountId:string*: Indirizzo email dell'autore originale.
- *+companyId:string*: Codice univoco dell'azienda al quale il database appartiene.
- *+databaseId:string*: Codice univoco del database al quale attingere per i dati.

Metodi

- *+runquery(id:string,data:Object,cb:function*: Esegue una query NoSQL sul database assegnato alla DSLI.

Hooks

- *BeforeRunQuery*: Eseguire le query necessarie ad una DSLI è permesso a chiunque disponga del rispettivo permesso di esecuzione.

Relazioni con altre classi

Le relazioni seguenti sono di fatto riferimenti fra modelli di dati
BackEnd :: Common :: Models :: Account - l'utente autore
BackEnd :: Common :: Models :: Company - l'azienda di appartenenza
BackEnd :: Common :: Models :: Database - il database a cui la DSLI fa riferimento

3.3.5 BackEnd :: Common :: Models :: Duty

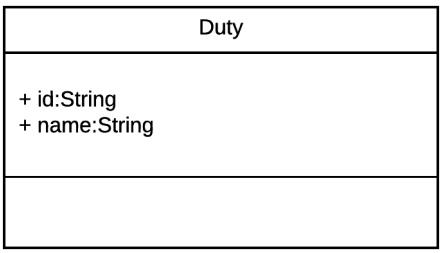

Fig 5: Duty

Descrizione

Classe che raccoglie tutti i possibili ruoli ricopribili all'interno dell'azienda.

Utilizzo

Questa classe estende dalla classe base loopback PersistedModel.

Attributi

- *+Id:string*: Codice univoco.
- *+name:string*: Nome mnemonico per distinguere i ruoli fra loro.

3.3.6 BackEnd :: Common :: MethodDisabler

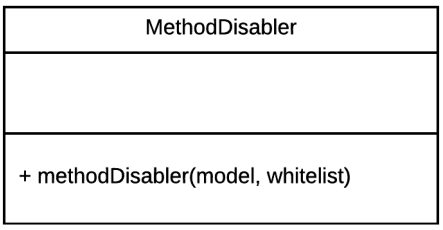

Fig 6: MethodDisabling

Descrizione

Classe contenente una funzione statica per disabilitare le API non desiderate.

Utilizzo

Chiama il metodo loopback disableRemoteMethod su tutte le API inserite nella *white-list*$_g$.

4 Specifica classi del FrontEnd

4.1 Actions

Questo package contiene tutti gli action creator del front-end. Ogni classe è composta da tre action creator dei quali uno, il principale è pubblico mentre i due rimanenti sono privati; in altre parole è possibile chiamare solo l'action creator principale al di fuori della classe mentre gli altri due vengono chiamati esclusivamente dall'action creator pubblico. I tre action creator hanno tre compiti differenti:

- *request*: questa action creator ha il compito di lanciare una semplice action di tipo 'waiting' con un campo stringa apposito con il compito di descrivere l'azione per cui si sta attendendo l'esito;
- *receive*: questa action creator, a seconda di un parametro di tipo booleano fornito, genera due diverse action:
  1. se la variabile booleana è positiva viene emanata una action con tipo corrispondente al nome dell'operazione (se l'action creator changeImage termina con esito positivo verrà lanciata una action di tipo 'changeImage'). Dipendentemente dalle operazioni svolte all'interno dell'action creator è possibile che venga inserito all'interno della action anche l'oggetto restituito dalla richiesta API;
  2. se la variabile booleana è positiva viene emanata una action di errore, ovvero una action di tipo 'error' contenente l'oggetto descrivente l'errore ottenuto in seguito alla richiesta API.
- *action creator pubblica*: questa action creator è sempre asincrona, ovvero non ritorna una action, un semplice oggetto, ma ritorna una funzione. Il comportamento di questi action creator è generalmente il seguente:
  - Viene chiamata la request, con il compito di comunicare lo stato d'attesa al client;
  - Viene effettuata una qualche richiesta API;
  - In base all'esito della richiesta vengono passati valori diversi alla receive, che comunicherà l'esito della richiesta al client.

4.1.1 FrontEnd :: Actions :: rootAction

Descrizione

Classe che ha come compito quello di dare un semplice accesso ai vari action creators.

Utilizzo

Questa classe mette a disposizione i vari action creator importandoli al proprio interno e contemporaneamente esportandoli verso l'esterno.

Relazioni con altre classi

Le relazioni seguenti esistono con il puro scopo di centralizzare l'accesso ad esse, non vengono quindi utilizzate internamente a questa classe.

FrontEnd::Actions::companyRegistration
FrontEnd::Actions::userRegistration
FrontEnd::Actions::login
FrontEnd::Actions::emailResetPassword
FrontEnd::Actions::redirect
FrontEnd::Actions::changePassword
FrontEnd::Actions::changeImage
FrontEnd::Actions::execDSLI
FrontEnd::Actions::newDSLI
FrontEnd::Actions::deleteDSLI
FrontEnd::Actions::cloneDSLI
FrontEnd::Actions::saveTextDSLI
FrontEnd::Actions::resetTextDSLI
FrontEnd::Actions::inviteNewUser
FrontEnd::Actions::changeAccessLevel
FrontEnd::Actions::deleteUser
FrontEnd::Actions::changeDSLIPermits
FrontEnd::Actions::embodyUser
FrontEnd::Actions::contactSupport
FrontEnd::Actions::getDSLI
FrontEnd::Actions::getDSLIList
FrontEnd::Actions::addDatabase

Metodi

Questa classe ha il compito di offrire un accesso centralizzato ai vari action creator definiti. Al suo interno vengono importati i vari action creator e vengono in seguito esportati verso l'esterno, ciò permette di poter chiamare qualunque action creator importando esclusivamente questa classe. In altre parole questa classe non è altro che un centro di smistamento verso i vari action creator, di conseguenza non possiede metodi propri.

4.1.2 FrontEnd :: Actions :: companyRegistration

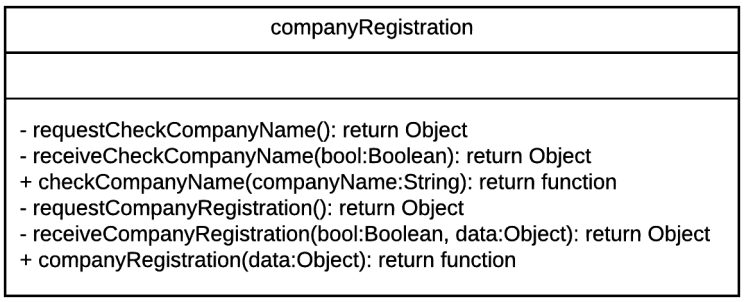

Fig 7: companyRegistration

Descrizione

Action creator che ha il compito di registrare un'azienda all'interno del sistema.

Utilizzo

Questa action creator ha il compito di verificare la validità dei dati inseriti e, se i dati sono validi, manda una richiesta di inserimento dati al sistema e genera una action contente le nuove informazioni, altrimenti genera una action di errore.

Relazioni con altre classi

FrontEnd::Actions::redirect: redirect viene utilizzato per permettere a questa action creator di reindirizzare l'utente ad operazione avvenuta.
FrontEnd::Actions::userRegistration: se la registrazione dell'azienda va a buon fine essa termina chiamando la procedura per la registrazione del proprietario.

Metodi

- *-requestCheckCompanyName()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;
- *-receiveCheckCompanyName(bool:Boolean)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'checkCompanyName' che descrive la corretta esecuzione di checkCompanyName o una action di tipo 'failedCheckCompanyName';
- *+checkCompanyName(companyName:String)*: funzione con il compito di effettuare una richiesta *GET companies/{id}/exists* al server per verificare l'unicità del campo companyName all'interno del database. Ciò avviene dopo aver chiamato requestCheckCompanyName per mostrare lo stato di attesa. Una volta che la richiesta all'API è stata soddisfatta viene chiamato l'action creator receiveCheckCompanyName a cui, in base all'esito della richiesta, verranno passate diverse informazioni;
- *-requestCompanyRegistration()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;

- *-receiveCompanyRegistration(bool:Boolean, data:Object)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'companyRegistration' che descrive la corretta esecuzione dell'action creator o una action di tipo 'error'. Il parametro data corrisponde all'esito della richiesta API che, in base al valore assunto da bool, corrisponderà alla voce inserita o alla descrizione dell'errore riscontrato;
- *+companyRegistration(data:Object)*: funzione con il compito di effettuare una richiesta *POST companies* al server per registrare l'azienda, le cui informazioni sono contenute all'interno del parametro 'data'. Ciò avviene dopo aver chiamato requestCompanyRegistration per mostrare lo stato di attesa. Una volta che la richiesta all'API è stata soddisfatta viene chiamato l'action creator receiveCompanyRegistration a cui, in base all'esito della richiesta, verranno passate diverse informazioni.

4.1.3 FrontEnd :: Actions :: userRegistration

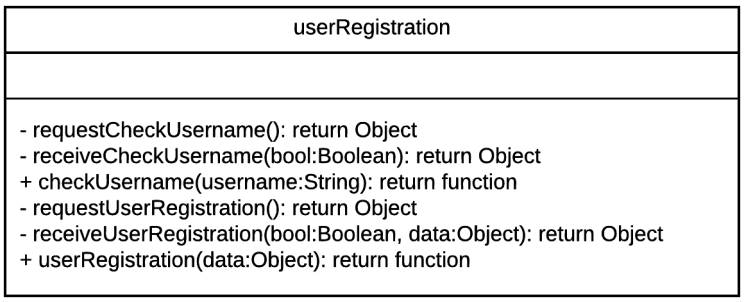

Fig 8: userRegistration

Descrizione

Action creator che ha il compito di registrare un utente all'interno del sistema.

Utilizzo

Questa action creator ha il compito di verificare la validità dei dati inseriti e, se i dati sono validi, manda una richiesta di inserimento dati al sistema e genera una action contente le nuove informazioni, altrimenti genera una action di errore.

Relazioni con altre classi

FrontEnd::Actions::redirect: redirect viene utilizzato per permettere a questa action creator di reindirizzare l'utente ad operazione avvenuta.

Metodi

- *-requestCheckUsername()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;
- *-receiveCheckUsername(bool:Boolean)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'checkUsername' che descrive la corretta esecuzione di checkUsername o una action di tipo 'failedCheckUsername';
- *+checkUsername(username:String)*: funzione con il compito di effettuare una richiesta *GET accounts/{id}/exists* al server per verificare l'unicità del campo username all'interno del database. Ciò avviene dopo aver chiamato requestCheckUsername per mostrare lo stato di attesa. Una volta che la richiesta all'API è stata soddisfatta viene chiamato l'action creator receiveCheckUsername a cui, in base all'esito della richiesta, verranno passate diverse informazioni;
- *-requestUserRegistration()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;
- *-receiveUserRegistration(bool:Boolean, data:Object)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'userRegistration' che descrive la corretta esecuzione

dell'action creator o una action di tipo 'error'. Il parametro data corrisponde all'esito della richiesta API che, in base al valore assunto da bool, corrisponderà alla voce inserita o alla descrizione dell'errore riscontrato;

- *+userRegistration(data:Object)*: funzione con il compito di effettuare una richiesta *POST accounts* al server per registrare un nuovo utente, le cui informazioni sono contenute all'interno del parametro 'data'. Ciò avviene dopo aver chiamato requestUserRegistration per mostrare lo stato di attesa. Una volta che la richiesta all'API è stata soddisfatta viene chiamato l'action creator receiveUserRegistration a cui, in base all'esito della richiesta, verranno passate diverse informazioni.

4.1.4 FrontEnd :: Actions :: login

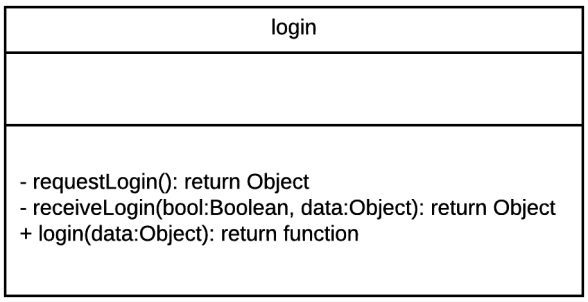

Fig 9: login

Descrizione

Action creator che ha il compito di autenticare un utente all'interno del sistema.

Utilizzo

Questa action creator ha il compito di verificare la validità dei dati inseriti mediante una richiesta al sistema e, se i dati sono validi, contatta il sistema per effettuare l'autenticazione e genera una action per descrivere l'operazione effettuata, altrimenti genera un'opportuna action d'errore.

Relazioni con altre classi

FrontEnd::Actions::redirect: redirect viene utilizzato per permettere a questa action creator di reindirizzare l'utente ad operazione avvenuta.

Metodi

- *-requestLogin()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;
- *-receiveLogin(bool:Boolean, data:Object)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'login' che descrive la corretta esecuzione dell'action creator o una action di tipo 'error'. Il parametro data corrisponde all'esito della richiesta API che, in base al valore assunto da bool, corrisponderà ai dati dell'utente o alla descrizione dell'errore riscontrato;
- *+login(data:Object)*: funzione con il compito di effettuare una richiesta *POST accounts/login?include=user* al server per autenticare l'utente, le cui informazioni sono contenute all'interno del parametro 'data'. Ciò avviene dopo aver chiamato requestLogin per mostrare lo stato di attesa. Una volta che la richiesta all'API è stata soddisfatta viene chiamato l'action creator receiveLogin a cui, in base all'esito della richiesta, verranno passate diverse informazioni.

4.1.5 FrontEnd :: Actions :: emailResetPassword

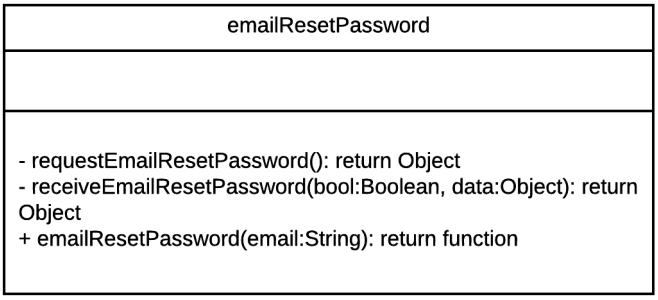

Fig 10: emailResetPassword

Descrizione

Action creator che ha il compito di iniziare la procedura di reset password.

Utilizzo

Questa action creator ha il compito di verificare la validità dei dati inseriti mediante una richiesta al sistema e, se i dati sono validi, effettua una richiesta a GET accounts/id/-pwdmail e genera una action per descrivere l'operazione compiuta, altrimenti genera un'opportuna action d'errore.

Relazioni con altre classi

FrontEnd::Actions::redirect: redirect viene utilizzato per permettere a questa action creator di reindirizzare l'utente ad operazione avvenuta.

Metodi

- *-requestResetMail()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;
- *-receiveResetMail(bool:Boolean, data:Object)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'sendResetMail' che descrive la corretta esecuzione dell'action creator o una action di tipo 'error'. Il parametro data corrisponde all'esito della richiesta API che, in base al valore assunto da bool, corrisponderà alle informazioni inserite o alla descrizione dell'errore riscontrato;
- *+sendResetMail(email:String)*: funzione con il compito di effettuare una richiesta *POST accounts/{id}/pwdmail* al server per richiedere l'invio dell'email per il reset della password, l'email a cui deve essere spedito il link per il reset è identificata dal parametro 'email'. Ciò avviene dopo aver chiamato requestResetMail per mostrare lo stato di attesa. Una volta che la richiesta all'API è stata soddisfatta viene chiamato l'action creator receiveResetMail a cui, in base all'esito della richiesta, verranno passate diverse informazioni.

4.1.6 FrontEnd :: Actions :: redirect

Descrizione

Action creator che ha il compito di far reindirizzare la pagina.

Utilizzo

Questa action creator ha il compito di generare una action contente la pagina a cui si deve venire reindirizzati.

4.1.7 FrontEnd :: Actions :: changePassword

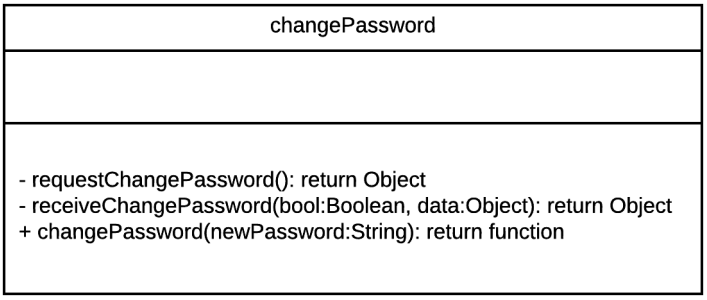

Fig 11: changePassword

Descrizione

Action creator che ha il compito di far modificare la password dell'utente.

Utilizzo

Questa action creator ha il compito di mandare una richiesta di aggiornamento password, corredata di un token specifico per autorizzare l'operazione.

Relazioni con altre classi

FrontEnd::Actions::redirect: redirect viene utilizzato per permettere a questa action creator di reindirizzare l'utente ad operazione avvenuta.

Metodi

- *-requestChangePassword()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;
- *-receiveChangePassword(bool:Boolean, data:Object)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'changePassword' che descrive la corretta esecuzione dell'action creator o una action di tipo 'error'. Il parametro data corrisponde all'esito della richiesta API nel caso sia riscontrato qualche errore;
- *+changePassword(newPassword:String)*: funzione con il compito di effettuare una richiesta *POST accounts/{id}/newpwd* al server per modificare la password, passata tramite il parametro 'newPassword'. Ciò avviene dopo aver chiamato requestChangePassword per mostrare lo stato di attesa. Una volta che la richiesta all'API è stata soddisfatta viene chiamato l'action creator receiveChangePassword a cui, in base all'esito della richiesta, verranno passate diverse informazioni.

4.1.8 FrontEnd :: Actions :: execDSLI

Descrizione

Action creator che ha il compito di far eseguire il codice della DSLI.

Utilizzo

Questa action creator ha il compito di effettuare una chiamata al sistema per ottenere l'esito dell'esecuzione della DSLI selezionata. In caso non ci siano problemi nei dati ottenuti la classe genera una action contente tali dati, altrimenti genera una action di errore.

Relazioni con altre classi

FrontEnd::Actions::redirect
FrontEnd::Services::PageBuilder

4.1.9 FrontEnd :: Actions :: newDSLI

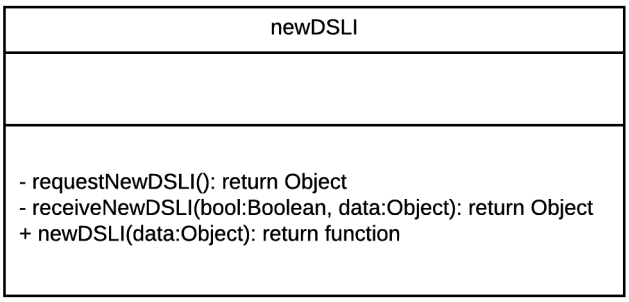

Fig 12: newDSLI

Descrizione

Action creator che ha il compito di far aggiungere una nuova DSLI.

Utilizzo

Questa action creator ha il compito di mandare una richiesta per l'inserimento della nuova DSLI al sistema e generare una action contente il nome della nuova DSLI.

Relazioni con altre classi

FrontEnd::Actions::redirect: redirect viene utilizzato per permettere a questa action creator di reindirizzare l'utente ad operazione avvenuta.

Metodi

- *-requestNewDSLI()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;
- *-receiveNewDSLI(bool:Boolean, data:Object)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'newDSLI' che descrive la corretta esecuzione dell'action creator o una action di tipo 'error'. Il parametro data corrisponde all'esito della richiesta API che, in base al valore assunto da bool, corrisponderà alla dsli corredata dalle informazioni del database o alla descrizione dell'errore riscontrato;
- *+newDSLI(data:Object)*: funzione con il compito di effettuare una richiesta *POST companies/{id}/dsls* al server per creare una nuova DSLI, le cui informazioni vengono passate tramite il parametro 'data'. Ciò avviene dopo aver chiamato requestNewDSLI per mostrare lo stato di attesa. Una volta che la richiesta all'API è stata soddisfatta viene chiamato l'action creator receiveNewDSLI a cui, in base all'esito della richiesta, verranno passate diverse informazioni.

4.1.10 FrontEnd :: Actions :: deleteDSLI

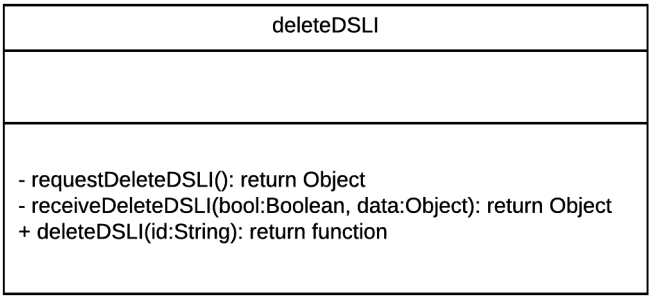

Fig 13: deleteDSLI

Descrizione

Action creator che ha il compito di far cancellare una DSLI dal sistema.

Utilizzo

Questa action creator ha il compito di mandare una richiesta di rimozione per la DSLI selezionata al sistema e generare una action che descrive l'operazione effettuata.

Relazioni con altre classi

FrontEnd::Actions::redirect: redirect viene utilizzato per permettere a questa action creator di reindirizzare l'utente ad operazione avvenuta.

Metodi

- *-requestDeleteDSLI()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;
- *-receiveDeleteDSLI(bool:Boolean, data:Object)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'deleteDSLI' che descrive la corretta esecuzione dell'action creator o una action di tipo 'error'. Il parametro data corrisponde all'esito della richiesta API nel caso sia riscontrato qualche errore;
- *+deleteDSLI(id:String)*: funzione con il compito di effettuare una richiesta *DELETE companies/{company_id}/dsls/{dsli_id}* al server per eliminare una DSLI, identificata tramite il parametro 'id'. Ciò avviene dopo aver chiamato requestDeleteDSLI per mostrare lo stato di attesa. Una volta che la richiesta all'API è stata soddisfatta viene chiamato l'action creator receiveDeleteDSLI a cui, in base all'esito della richiesta, verranno passate diverse informazioni.

4.1.11 FrontEnd :: Actions :: cloneDSLI

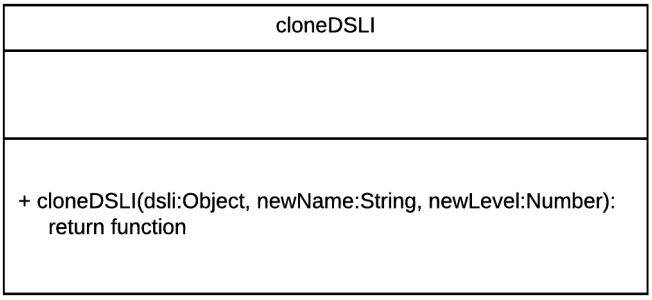

Fig 14: cloneDSLI

Descrizione

Action creator che ha il compito di far clonare una DSLI.

Utilizzo

Questa action creator ha il compito di mandare una richiesta per l'inserimento della nuova DSLI al sistema e generare una action contente il nome della nuova DSLI.

Relazioni con altre classi

FrontEnd::Actions::redirect: redirect viene utilizzato per permettere a questa action creator di reindirizzare l'utente ad operazione avvenuta.

Metodi

- *+cloneDSLI(dsli:Object, newName:String, newLevel:Number)*: funzione con il compito di creare una nuova DSLI, copiando una DSLI esistente, fornita tramite il parametro 'dsli', usando come nome quello fornito tramite il parametro 'newName' e cambiando il livello d'accesso impostando come nuovo valore quello passato tramite il parametro 'newLevel'. Ciò avviene modificando le informazioni contenute in 'dsli' con quanto contenuto negli altri due parametri; effettuate le modifiche verrà passato l'oggetto modificato alla action creator *newDSLI*.

4.1.12 FrontEnd :: Actions :: saveTextDSLI

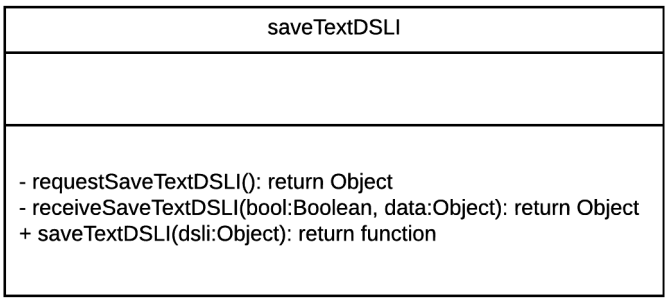

Fig 15: saveTextDSLI

Descrizione

Action creator che ha il compito di far salvare il testo modificato di una DSLI.

Utilizzo

Questa action creator ha il compito di mandare una richiesta di aggiornamento dati al sistema e genera una action contente le informazioni modificate.

Relazioni con altre classi

FrontEnd::Actions::redirect: redirect viene utilizzato per permettere a questa action creator di reindirizzare l'utente ad operazione avvenuta.

Metodi

- *-requestSaveTextDSLI()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;
- *-receiveSaveTextDSLI(bool:Boolean, data:Object)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'saveTextDSLI' che descrive la corretta esecuzione dell'action creator o una action di tipo 'error'. Il parametro data corrisponde all'esito della richiesta API che, in base al valore assunto da bool, corrisponderà alle informazioni aggiornate della dsli o alla descrizione dell'errore riscontrato;
- *+saveTextDSLI(dsli:Object)*: funzione con il compito di effettuare una richiesta *PUT companies/{company_id}/dsls/{dsli_id}* al server per sostituire il codice di una DSLI, identificata tramite il parametro 'id', usando come nuovo codice quello fornito tramite il parametro 'newName'. Ciò avviene dopo aver chiamato requestSaveTextDSLI per mostrare lo stato di attesa. Una volta che la richiesta all'API è stata soddisfatta viene chiamato l'action creator receiveSaveTextDSLI a cui, in base all'esito della richiesta, verranno passate diverse informazioni.

4.1.13 FrontEnd :: Actions :: setAccessLevel

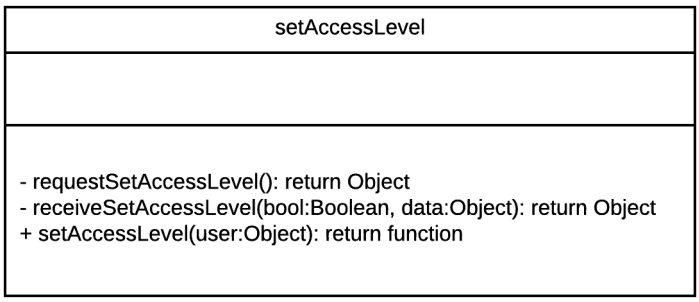

Fig 16: setAccessLevel

Descrizione

Action creator che ha il compito di far modificare il livello d'accesso di un utente.

Utilizzo

Questa action creator ha il compito di mandare una richiesta di aggiornamento per il livello d'accesso dell'utente al sistema e genera una action contente le informazioni modificate.

Relazioni con altre classi

FrontEnd::Actions::redirect: redirect viene utilizzato per permettere a questa action creator di reindirizzare l'utente ad operazione avvenuta.

Metodi

- *-requestSetAccessLevel()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;
- *-receiveSetAccessLevel(bool:Boolean, data:Object)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'changeAccessLevel' che descrive la corretta esecuzione dell'action creator o una action di tipo 'error'. Il parametro data corrisponde all'esito della richiesta API nel caso sia riscontrato qualche errore;
- *+setAccessLevel(user:Object)*: funzione con il compito di effettuare una richiesta *POST companies/{company_id}/users/{user_id}/permit/{newLevel}* al server per modificare il livello d'accesso dell'utente autenticato, il cui nuovo valore viene passato tramite il parametro 'newLevel'. Ciò avviene dopo aver chiamato requestChangeAccessLevel per mostrare lo stato di attesa. Una volta che la richiesta all'API è stata soddisfatta viene chiamato l'action creator receiveChangeAccessLevel a cui, in base all'esito della richiesta, verranno passate diverse informazioni.

4.1.14 FrontEnd :: Actions :: deleteUser

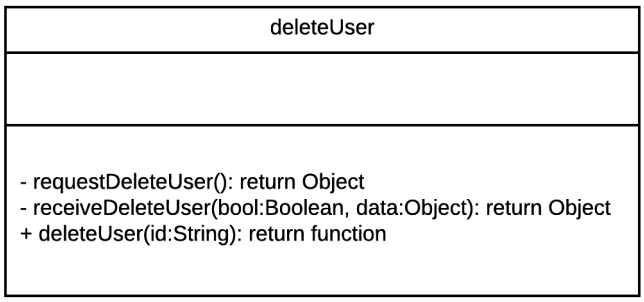

Fig 17: deleteUser

Descrizione

Action creator che ha il compito di far eliminare un utente dal sistema,

Utilizzo

Questa action creator ha il compito di mandare una richiesta di rimozione per l'utente selezionato al sistema e generare una action che descrive l'operazione effettuata.

Relazioni con altre classi

FrontEnd::Actions::redirect: redirect viene utilizzato per permettere a questa action creator di reindirizzare l'utente ad operazione avvenuta.

Metodi

- *-requestDeleteUser()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;
- *-receiveDeleteUser(bool:Boolean, data:Object)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'deleteUser' che descrive la corretta esecuzione dell'action creator o una action di tipo 'error'. Il parametro data corrisponde all'esito della richiesta API nel caso sia riscontrato qualche errore;
- *+deleteUser(id:String)*: funzione con il compito di effettuare una richiesta *DELETE companies/{company_id}/users/{user_id}* al server per eliminare un utente, identificato tramite il parametro 'id'. Ciò avviene dopo aver chiamato requestDeleteUser per mostrare lo stato di attesa. Una volta che la richiesta all'API è stata soddisfatta viene chiamato l'action creator receiveDeleteUser a cui, in base all'esito della richiesta, verranno passate diverse informazioni.

4.1.15 FrontEnd :: Actions :: changeDSLIPermits

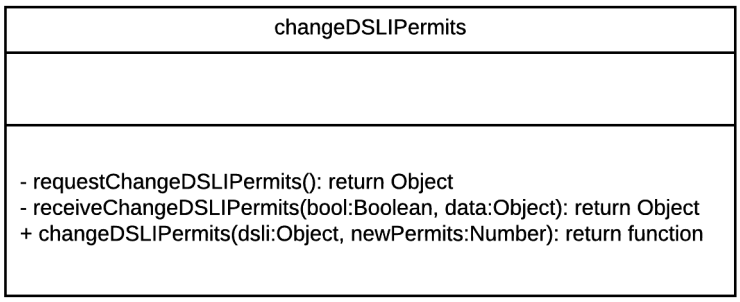

Fig 18: changeDSLIPermits

Descrizione

Action creator che ha il compito di far modificare i diritti d'accesso ad una DSLI.

Utilizzo

Questa action creator ha il compito di mandare una richiesta di aggiornamento per i diritti d'accesso alla DSLI selezionata al sistema e genera una action contente le informazioni modificate.

Relazioni con altre classi

FrontEnd::Actions::redirect: redirect viene utilizzato per permettere a questa action creator di reindirizzare l'utente ad operazione avvenuta.

Metodi

- *-requestChangeDSLIPermits()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;
- *-receiveChangeDSLIPermits(bool:Boolean, data:Object)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'changeDSLIPermits' che descrive la corretta esecuzione dell'action creator o una action di tipo 'error'. Il parametro data corrisponde all'esito della richiesta API che, in base al valore assunto da bool, corrisponderà al nuovo permesso d'accesso o alla descrizione dell'errore riscontrato;
- *+changeDSLIPermits(dsli:Object, newPermits:Number)*: funzione con il compito di effettuare una richiesta *PUT companies/{company_id}/dsls/{dsli_id}* al server per modificare i permessi d'accesso ad una DSLI, i cui nuovi valori vengono passati tramite il parametro 'newPermits'. Ciò avviene dopo aver chiamato requestChangeDSLIPermits per mostrare lo stato di attesa. Una volta che la richiesta all'API è stata soddisfatta viene chiamato l'action creator receiveChangeDSLIPermits a cui, in base all'esito della richiesta, verranno passate diverse informazioni.

4.1.16 FrontEnd :: Actions :: embodyUser

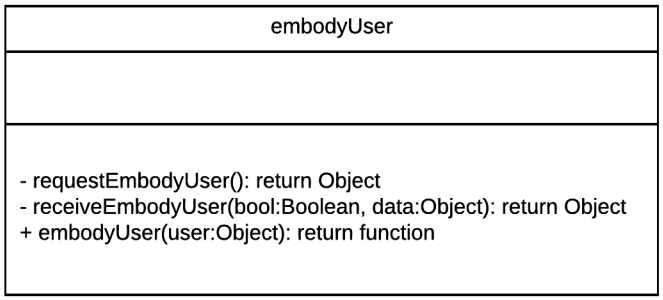

Fig 19: embodyUser

Descrizione

Action creator che ha il compito di far impersonare al superadmin un utente di una data azienda all'interno del sistema.

Utilizzo

Questa action creator ha il compito di verificare la validità dei dati inseriti mediante una richiesta al sistema e, se i dati sono validi, contatta il sistema per effettuare l'impersonificazione e genera una action per descrivere l'operazione effettuata, altrimenti genera un'opportuna action d'errore.

Relazioni con altre classi

FrontEnd::Actions::redirect: redirect viene utilizzato per permettere a questa action creator di reindirizzare l'utente ad operazione avvenuta.

Metodi

- *-requestEmbodyUser()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;
- *-receiveEmbodyUser(bool:Boolean, data:Object)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'embodyUser' che descrive la corretta esecuzione dell'action creator o una action di tipo 'error'. Il parametro data corrisponde all'esito della richiesta API che, in base al valore assunto da bool, corrisponderà alle informazioni dell'utente impersonato o alla descrizione dell'errore riscontrato;
- *+embodyUser(data:Object)*: funzione con il compito di permettere al super-admin di impersonare un qualsiasi utente registrato al sistema, le cui informazioni minime vengono fornite tramite il parametro 'data'. Ciò avviene dopo aver chiamato requestEmbodyUser per mostrare lo stato di attesa. Una volta che la richiesta *GET accounts/{account_id}* è stata soddisfatta viene chiamato l'action creator receiveEmbodyUser a cui, in base all'esito della richiesta, verranno passate diverse informazioni.

4.1.17 FrontEnd :: Actions :: contactSupport

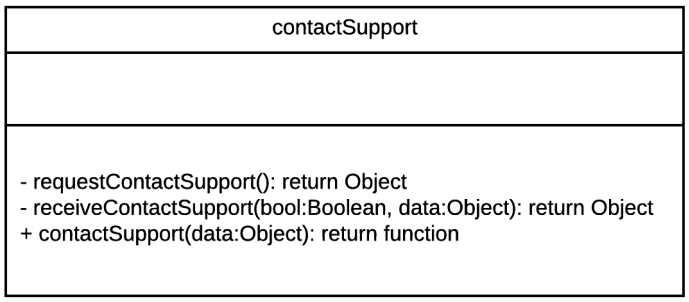

Fig 20: contactSupport

Descrizione

Action creator che ha il compito di recapitare al supporto del sito un messaggio dell'utente.

Utilizzo

Questa action creator ha il compito di effettuare una chiamata al server remoto e generare una action che descrive l'operazione effettuata.

Relazioni con altre classi

FrontEnd::Actions::redirect: redirect viene utilizzato per permettere a questa action creator di reindirizzare l'utente ad operazione avvenuta.

Metodi

- *-requestContactSupport()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;
- *-receiveContactSupport(bool:Boolean, data:Object)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'contactSupport' che descrive la corretta esecuzione dell'action creator o una action di tipo 'error'. Il parametro data corrisponde all'esito della richiesta API nel caso sia riscontrato qualche errore;
- *+contactSupport(data:Object)*: funzione con il compito di effettuare una richiesta *POST accounts/help/{id}* al server per inviare una mail al supporto di MaaS, dove oggetto e testo dell'email vengono passati tramite il parametro 'data'. Ciò avviene dopo aver chiamato requestContactSupport per mostrare lo stato di attesa. Una volta che la richiesta all'API è stata soddisfatta viene chiamato l'action creator receiveContactSupport a cui, in base all'esito della richiesta, verranno passate diverse informazioni.

4.1.18 FrontEnd :: Actions :: getDSLI

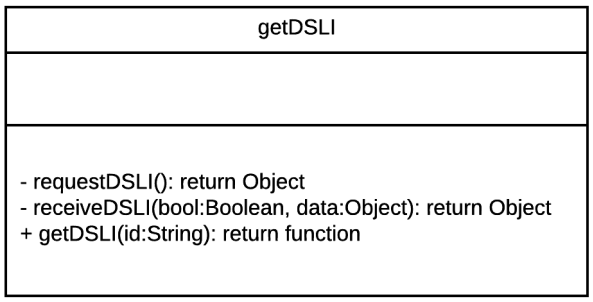

Fig 21: getDSLI

Descrizione

Action creator che ha il compito di prendere le informazioni di una DSLI dal sistema.

Utilizzo

Questa action creator ha il compito di mandare una richiesta all'API per ottenere le informazioni complete relative ad una DSLI determinata tramite il proprio id.

Relazioni con altre classi

FrontEnd::Actions::redirect: redirect viene utilizzato per permettere a questa action creator di reindirizzare l'utente ad operazione avvenuta.

Metodi

- *-requestDSLI()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;
- *-receiveDSLI(bool:Boolean, data:Object)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'getDSLI' che descrive la corretta esecuzione dell'action creator o una action di tipo 'error'. Il parametro data corrisponde all'esito della richiesta API che, in base al valore assunto da bool, corrisponderà alle informazioni della dsli desiderata o alla descrizione dell'errore riscontrato;
- *+getDSLI(id:String)*: funzione con il compito di effettuare una richiesta *GET companies/{company_id}/dsls/{dsli_id}* al server per ottenere le informazioni relative ad una DSLI identificata tramite il parametro 'id'. Ciò avviene dopo aver chiamato requestDSLI per mostrare lo stato di attesa. Una volta che la richiesta all'API è stata soddisfatta viene chiamato l'action creator receiveDSLI a cui, in base all'esito della richiesta, verranno passate diverse informazioni.

4.1.19 FrontEnd :: Actions :: getDSLIList

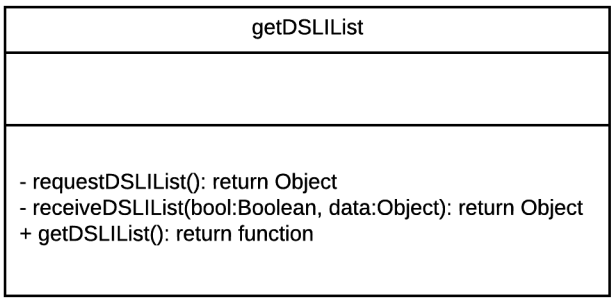

Fig 22: getDSLIList

Descrizione

Action creator che ha il compito di prendere dal sistema la lista delle DSLI accessibili dall'utente.

Utilizzo

Questo action creator ha il compito di mandare una richiesta all'API per ottenere la lista delle DSLI personali dell'utente e di quelle pubbliche all'interno dell'azienda.

Relazioni con altre classi

FrontEnd::Actions::redirect: redirect viene utilizzato per permettere a questa action creator di reindirizzare l'utente ad operazione avvenuta.

Metodi

- *-requestDSLIList()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;
- *-receiveDSLIList(bool:Boolean, data:Object)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'getDSLIList' che descrive la corretta esecuzione dell'action creator o una action di tipo 'error'. Il parametro data corrisponde all'esito della richiesta API che, in base al valore assunto da bool, corrisponderà alla lista delle dsli accessibili all'utente o alla descrizione dell'errore riscontrato;
- *+getDSLIList()*: funzione con il compito di effettuare una richiesta *GET companies/{company_id}/dsls* al server per ottenere la lista delle DSLI accessibili dall'utente. Ciò avviene dopo aver chiamato requestDSLIList per mostrare lo stato di attesa. Una volta che la richiesta all'API è stata soddisfatta viene chiamato l'action creator receiveDSLIList a cui, in base all'esito della richiesta, verranno passate diverse informazioni.

4.1.20 FrontEnd :: Actions :: addDatabase

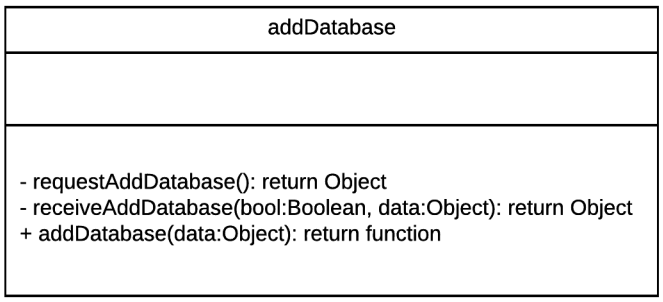

Fig 23: addDatabase

Descrizione

Action creator che ha il compito di memorizzare i database di un'azienda all'interno del sistema.

Utilizzo

Questa action creator ha il compito di mandare una richiesta per l'inserimento delle stringhe d'accesso ai database dell'azienda al sistema e genera una action contente le nuove informazioni, altrimenti genera una action di errore.

Metodi

- *-requestAddDatabase()*: funzione con il compito di creare una action di tipo 'waiting' che descrive l'operazione in attesa di completamento;
- *-receiveAddDatabase(bool:Boolean, data:Object)*: funzione che in base al valore di bool, che può essere equivalente a 'true' o 'false', restituisce rispettivamente una action di tipo 'addDatabase' che descrive la corretta esecuzione dell'action creator o una action di tipo 'error'. Il parametro data corrisponde all'esito della richiesta API nel caso sia riscontrato qualche errore;
- *+addDatabase(data:Object)*: funzione con il compito di effettuare una richiesta *PUT companies/{company_id}/databases* al server per salvare la lista delle stringhe d'accesso ai database dell'azienda. Ciò avviene dopo aver chiamato requestAddDatabase per mostrare lo stato di attesa. Una volta che la richiesta all'API è stata soddisfatta viene chiamato l'action creator receiveAddDatabase a cui, in base all'esito della richiesta, verranno passate diverse informazioni.

4.2 Reducers

Questo package contiene tutti i reducers del front-end. Dato che i reducers non sono effettivamente delle classi è stato deciso di adottare il seguente formalismo:

- come **attributi** sono stati messi i campi dello stato dello store redux che, per come è stato organizzato il progetto sono sempre gestiti da un solo reducer. Ricordiamo che lo stato, e le voci che lo compongono, sono accessibili da qualunque punto del front-end, il che significa che tutti gli attributi sono sempre pubblici;
- un reducer, non essendo una classe, non possiede dei metodi interni ma è composto da uno switch e tutti i casi interni. Per questo motivo al posto dei metodi sono descritti i vari **casi** del reducer atti a gestire le varie action in entrata.

4.2.1 FrontEnd :: Reducers :: rootReducer

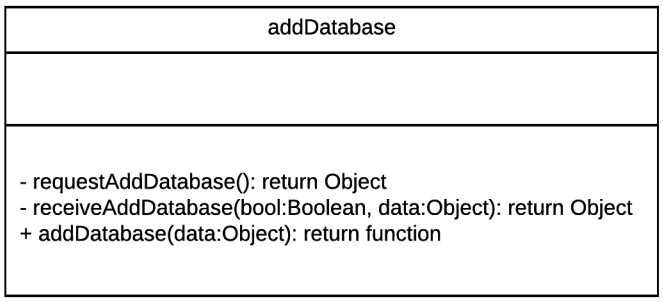

Fig 24: addDatabase

Descrizione

Reducer che gestisce l'accesso agli altri reducers.

Utilizzo

Questo reducer gestisce le action ricevute reindirizzandole all'apposito reducer.

Relazioni con altre classi

FrontEnd::Reducers::statusReducer
FrontEnd::Reducers::loggedUserReducer
FrontEnd::Reducers::currentDSLIReducer
FrontEnd::Reducers::DSLIListReducer
FrontEnd::Reducers::userListReducer
FrontEnd::Reducers::dataListReducer

4.2.2 FrontEnd :: Reducers :: statusReducer

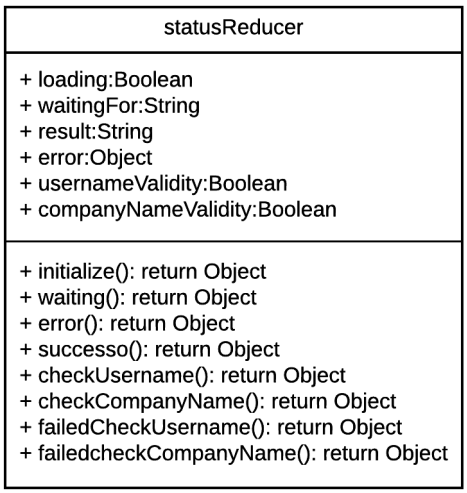

Fig 25: statusReducer

Descrizione
Reducer che gestisce lo status del sistema.

Utilizzo
Questo reducer gestisce le informazioni, ricevute tramite apposite action, relative alle operazioni avvenute nel sistema e aggiorna lo status in base all'azione effettuata.

Attributi

- *+loading:Boolean*: in base al valore assunto mostra se esiste una richiesta non ancora risolta o meno;
- *+waitingFor:String*: se loading è impostato a 'true' questo attributo contiene una stringa che definisce la richiesta non ancora soddisfatta;
- *+result:String*: definisce l'esito dell'ultima operazione, è può assumere i valori 'success' o 'error';
- *+error:Object*: se result assume il valore 'error' qui viene salvato l'errore restituito dall'API;
- *+usernameValidity:Boolean*: se siamo in una procedura di registrazione, questo attributo stabilisce se l'username fornito è disponibile o meno;
- *+companyNameValidity:Boolean*: se siamo nella procedura di registrazione di un'azienda, questo attributo stabilisce se il nome dell'azienda fornito è disponibile o meno.

Casi I reducer sono una semplice funzione che in base al tipo di azione ricevuta effettua operazioni differenti. Ora descriveremo i vari casi.

- *'initialize'*: imposta lo stato in una combinazione neutra;
- *'waiting'*: modifica lo stato mettendolo in condizione d'attesa;
- *'error'*: rimuove lo stato dalla condizione d'attesa e lo modifica per mostrare l'errore ricevuto;
- *successo*: questo caso, composto da tutte i casi di esito positivo, modifica lo stato per mostrare la corretta esecuzione della richiesta attesa;
- *checkUsername*: comportamento analogo al precedente con la differenza di impostare il parametro 'usernameValidity' a true;
- *checkCompanyName*: comportamento analogo al precedente con la differenza di impostare il parametro 'companyNameValidity' a true;
- *failedCheckUsername*: modifica lo stato mostrando come la richiesta sia stata soddisfatta e pone il parametro 'usernameValidity' a false;
- *failedcheckCompanyName*: modifica lo stato mostrando come la richiesta sia stata soddisfatta e pone il parametro 'companyNameValidity' a false.

4.2.3 FrontEnd :: Reducers :: loggedUserReducer

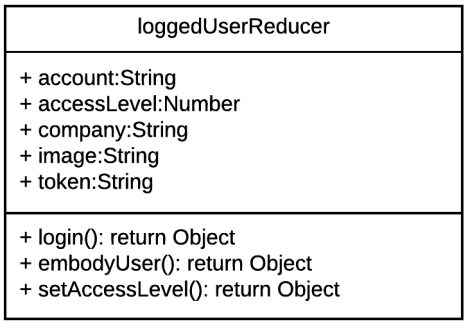

Fig 26: loggedUserReducer

Descrizione

Reducer che gestisce l'autenticazione al sistema e le informazioni dell'utente autenticato.

Utilizzo

Questo reducer gestisce le informazioni, ricevute tramite apposite action, relative all'autenticazione e alle informazioni dell'utente autenticato e genera un nuovo stato aggiornando il precedente con i dati ottenuti.

Attributi

- *+account:String*: attributo all'interno del quale viene memorizzata l'email dell'utente autenticato;
- *+accessLevel:Number*: attributo all'interno del quale viene memorizzato il livello d'accesso dell'utente autenticato;
- *+company:String*: attributo all'interno del quale viene memorizzato il nome dell'azienda dell'utente autenticato;
- *+image:String*: attributo all'interno del quale viene memorizzata l'immagine dell'utente autenticato;
- *+token:String*: attributo all'interno del quale viene memorizzato il token d'accesso alle risorse.

Casi I reducer sono una semplice funzione che in base al tipo di azione ricevuta effettua operazioni differenti. Ora descriveremo i vari casi.

- *'login, embodyUser'*: dopo che l'opportuna action creator ha correttamente fatto il login (o ha svolto l'impersonificazione) dell'utente nel sistema, vengono memorizzate le varie informazioni dell'utente, ottenute dall'apposito parametro della action;
- *'changeAccessLevel'*: modifica il livello d'acesso dell'utente nello stato;
- *'changeImage'*: modifica l'immagine dell'utente nello stato.

4.2.4 FrontEnd :: Reducers :: currentDSLIReducer

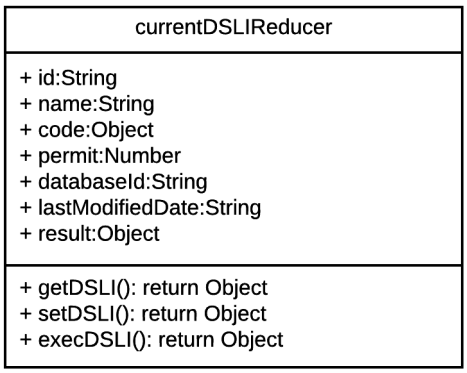

Fig 27: currentDSLIReducer

Descrizione
Reducer che gestisce la DSLI attualmente selezionata.

Utilizzo
Questo reducer gestisce le informazioni, ricevute tramite apposite action, relative alla DSLI correntemente selezionata permettendo di cambiare la DSLI selezionata e modificare le informazioni di tale DSLI.

Attributi

- *+id:String*: attributo all'interno del quale viene memorizzato l'id della DSLI correntemente selezionata;
- *+name:String*: attributo all'interno del quale viene memorizzato il nome della DSLI correntemente selezionata;
- *+code:Object*: attributo all'interno del quale viene memorizzato il codice della DSLI correntemente selezionata;
- *+permit:Number*: attributo all'interno del quale viene memorizzato il permesso dell'utente rispetto alla DSLI correntemente selezionata;
- *+databaseId:String*: attributo all'interno del quale viene memorizzato il database a cui è collegata la DSLI correntemente selezionata;
- *+lastModifiedDate:String*: attributo all'interno del quale viene memorizzata la data dell'ultima modifica alla DSLI correntemente selezionata;
- *+result:Object*: attributo all'interno del quale viene memorizzato l'esito dell'esecuzione della DSLI correntemente selezionata.

Casi I reducer sono una semplice funzione che in base al tipo di azione ricevuta effettua operazioni differenti. Ora descriveremo i vari casi.

- *'getDSLI, setDSLI'*: memorizza nello stato le informazioni della DSLI fornite alla action, rimuovendo eventuali informazioni precedentemente salvate;
- *'execDSLI'*: inserisce il risultato dell'esecuzione della DSLI nel campo result.

4.2.5 FrontEnd :: Reducers :: DSLIListReducer

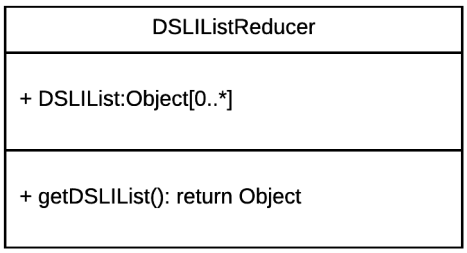

Fig 28: DSLIListReducer

Descrizione
Reducer che gestisce la lista delle DSLI disponibili per l'utente.

Utilizzo
Questo reducer gestisce le informazioni, ricevute tramite apposite action, relative alla lista delle DSLI disponibili permettendo di aggiornarla.

Attributi L'unico attributo di questo reducer è una lista di DSLI.

Casi I reducer sono una semplice funzione che in base al tipo di azione ricevuta effettua operazioni differenti. Ora descriveremo i vari casi.

- *'getDSLIList'*: memorizza nello stato la lista aggiornata delle DSLI accessibili dall'utente.

4.2.6 FrontEnd :: Reducers :: userListReducer

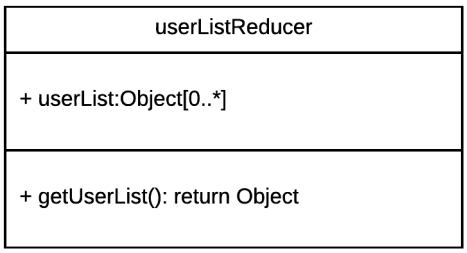

Fig 29: userListReducer

Descrizione
Reducer che gestisce la lista degli utenti registrati all'interno dell'azienda.

Utilizzo
Questo reducer gestisce le informazioni, ricevute tramite apposite action, relative alla lista degli utenti registrati permettendo di aggiornarla.

Attributi L'unico attributo di questo reducer è una lista di utenti.

Casi I reducer sono una semplice funzione che in base al tipo di azione ricevuta effettua operazioni differenti. Ora descriveremo i vari casi.

- *'getUserList'*: memorizza nello stato la lista aggiornata degli utenti iscritti all'azienda dell'utente.

4.2.7 FrontEnd :: Reducers :: dataListReducer

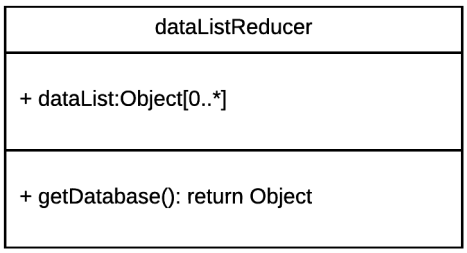

Fig 30: dataListReducer

Descrizione
Reducer che gestisce la lista dei database dell'azienda.

Utilizzo
Questo reducer gestisce le informazioni, ricevute tramite apposite action, relative alla lista dei database permettendo di aggiornarla.

Attributi L'unico attributo di questo reducer è una lista degli id dei database.

Casi I reducer sono una semplice funzione che in base al tipo di azione ricevuta effettua operazioni differenti. Ora descriveremo i vari casi.

- *'getUserList'*: memorizza nello stato la lista aggiornata dei database dell'azienda.

4.3 Components

4.3.1 FrontEnd ::View :: Components :: Components

Descrizione

File che raccoglie i riferimenti a tutti i Components creati dal team Matrioska.io.

Utilizzo

Grazie a questo file è possibile utilizzare i Components personalizzati in ogni parte del front-end grazie ad un solo import.

Relazioni con altre classi

FrontEnd :: View :: Components :: MTextBox
FrontEnd :: View :: Components :: MTextArea
FrontEnd :: View :: Components :: MUserRow
FrontEnd :: View :: Components :: MDSLIRow
FrontEnd :: View :: Components :: MAdminDSLIRow
FrontEnd :: View :: Components :: MDataRow
FrontEnd :: View :: Components :: MButton
FrontEnd :: View :: Components :: MError
FrontEnd :: View :: Components :: MLink

4.3.2 FrontEnd ::View :: Components :: MButton

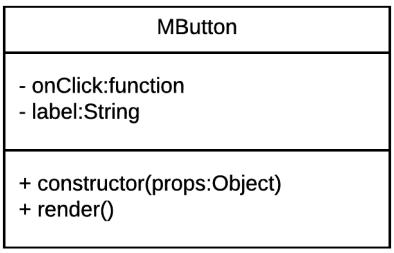

Fig 31: MButton

Descrizione

Componente configurabile rappresentante un pulsante.

Utilizzo

Creando un nuovo MButton si crea un bottone simile a quello html.

Attributi

- *-onClick:function*: Codice da eseguire quando il pulsante viene premuto;
- *-label:String*: Eventuale testo riportato su un pulsante.

4.3.3 FrontEnd ::View :: Components :: MDataRow

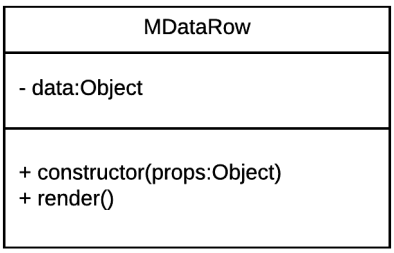

Fig 32: MDataRow

Descrizione

Componente rappresentante un database in una tabella.

Utilizzo

Quando è necessario ordinare in forma tabellare i database aziendali, questo componente rappresenta una riga visualizzando informazioni e offrendo funzionalità.

Attributi

- *-data:Object*: L'oggetto database completo dei parametri indispensabili.

Relazioni con altre classi

FrontEnd :: View :: Components :: MButton
FrontEnd::Actions::rootAction

4.3.4 FrontEnd ::View :: Components :: MDSLIRow

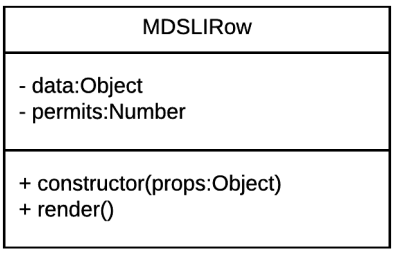

Fig 33: MDSLIRow

Descrizione

Componente rappresentante una DSLI in una tabella.

Utilizzo

Quando è necessario ordinare in forma tabellare delle DSLI, questo componente rappresenta una riga visualizzando informazioni e offrendo funzionalità.

Attributi

- *-data:Object*: L'oggetto DSLI completo dei parametri indispensabili;
- *-permits:Number*: Un valore che indica che tipo di permessi ha l'utente corrente sulla DSLI, così facendo si evita che un utente acceda a zone del sito web non utilizzabili.

Relazioni con altre classi

FrontEnd :: View :: Components :: MButton
FrontEnd::Actions::rootAction

4.3.5 FrontEnd ::View :: Components :: MAdminDSLIRow

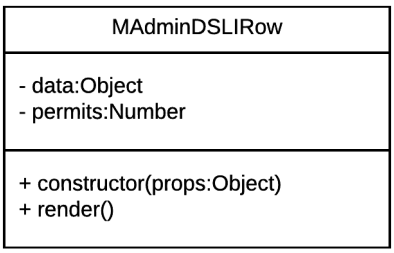

Fig 34: MAdminDSLIRow

Descrizione

Componente rappresentante una DSLI nella tabella per gli amministratori.

Utilizzo

Quando è necessario ordinare in forma tabellare delle DSLI, questo componente rappresenta una riga visualizzando informazioni e offrendo funzionalità amministratore.

Attributi

- *-data:Object*: L'oggetto DSLI completo dei parametri indispensabili;
- *-permits:Number*: Un valore che indica che tipo di permessi ha l'utente corrente sulla DSLI, così facendo si evita che un utente acceda a zone del sito web non utilizzabili.

Relazioni con altre classi

FrontEnd :: View :: Components :: MButton
FrontEnd::Actions::rootAction

4.3.6 FrontEnd ::View :: Components :: MUserRow

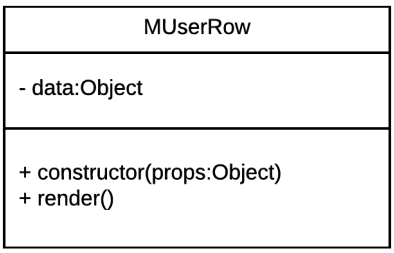

Fig 35: MUserRow

Descrizione

Componente rappresentante un utente in una tabella.

Utilizzo

Quando è necessario ordinare in forma tabellare degli utenti, questo componente rappresenta una riga visualizzando informazioni e offrendo funzionalità.

Attributi

- *-data:Object*: L'oggetto utente completo dei parametri indispensabili.

Relazioni con altre classi

FrontEnd :: View :: Components :: MButton
FrontEnd::Actions::rootAction

4.3.7 FrontEnd ::View :: Components :: MDataRow

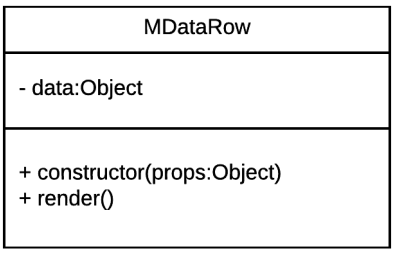

Fig 36: MDataRow

Descrizione

Componente rappresentante un database in una tabella.

Utilizzo

Quando è necessario ordinare in forma tabellare degli database, questo componente rappresenta una riga visualizzando informazioni e offrendo funzionalità.

Attributi

- *-data:Object*: L'oggetto database completo dei parametri indispensabili.

Relazioni con altre classi

FrontEnd :: View :: Components :: MButton
FrontEnd::Actions::rootAction

4.3.8 FrontEnd ::View :: Components :: MError

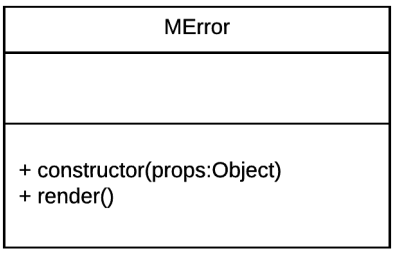

Fig 37: MError

Descrizione

Componente riportante eventuali errori registrati dal sistema.

Utilizzo

Questo componente può essere posizionato ovunque sia necessario notificare errori.

4.3.9 FrontEnd ::View :: Components :: MLink

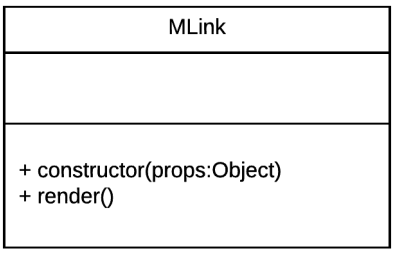

Fig 38: MLink

Descrizione

Semplice estensione di Link, fornisce un unico punto per modificare tutti i link del sito web.

Utilizzo

Questo componente eredita dalla classe Link di react-router-redux e consente di muoversi nel sito lanciando apposite action.

4.3.10 FrontEnd ::View :: Components :: MTextBox

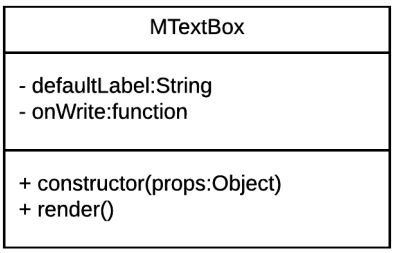

Fig 39: MTextBox

Descrizione

Componente configurabile rappresentante una casella di testo.

Utilizzo

Creando una nuova MTextBox si crea una casella di testo simile a quella html.

Attributi

- *-defaultLabel:String*: Definisce eventuale contenuto di deafult;
- *-onWrite:function*: Codice da eseguire quando il testo contenuto varia.

4.3.11 FrontEnd ::View :: Components :: MTextArea

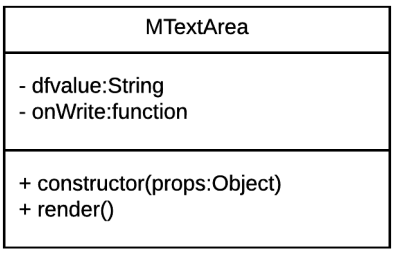

Fig 40: MTextArea

Descrizione

Questa classe è un componente rappresentante un'area di testo. Deve poter fornire il proprio valore in qualsiasi momento.

Utilizzo

Questo componente è utilizzato per chiedere agli utenti un valore testuale di elevate dimensioni e complessità.

Attributi

- *-dfvalue:String*: Definisce eventuale contenuto di deafult.
- *-onWrite:function*: Codice da eseguire quando il testo contenuto varia.

4.4 Containers

4.4.1 FrontEnd ::View :: Containers :: ContactSupport

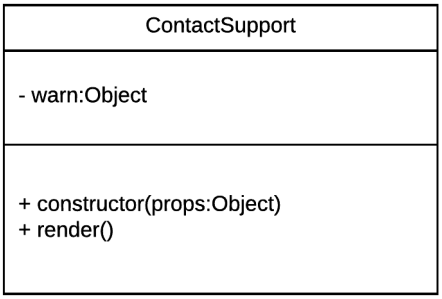

Fig 41: ContactSupport

Descrizione

Container che rappresenta la pagina web per il supporto clienti.

Utilizzo

La pagina dovrà visualizzare una MTextBox per l'indirizzo mail del cliente e una MTextArea per il messaggio. L'apposita action provvederà a spedire il messaggio tramite API.

Attributi

- *-warn:Object*: Messaggio di avvertimento per l'utente in caso i dati non siano validi.

Relazioni con altre classi

FrontEnd::View::Components
FrontEnd::Actions::rootAction

4.4.2 FrontEnd ::View :: Containers :: DataManagment

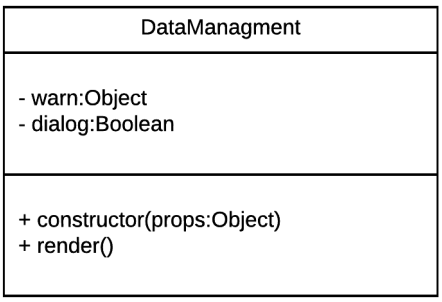

Fig 42: DataManagment

Descrizione

Container che rappresenta la pagina web per la gestione delle basi di dati.

Utilizzo

La pagina dovrà visualizzare una serie di MDataRow rappresentanti tutti i database aziendali. Sarà presente inoltre una coppia di MTextBox per aggiungerne di nuovi e una dialog per confermare eventuali eliminazioni.

Attributi

- *-warn:Object*: Messaggio di avvertimento per l'utente in caso i dati non siano validi;
- *-dialog:Boolean*: Valore booleano che indica se la dialog debba essere visibile.

Relazioni con altre classi

FrontEnd::View::Components
FrontEnd::Actions::rootAction

4.4.3 FrontEnd ::View :: Containers :: DSLIManagment

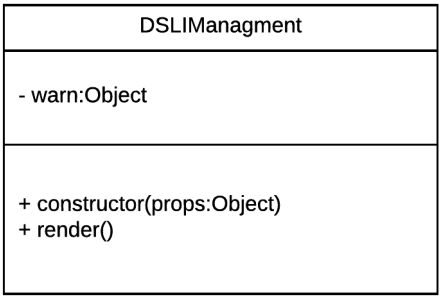

Fig 43: DSLIManagment

Descrizione

Container che rappresenta la pagina web per la gestione delle DSLI.

Utilizzo

La pagina dovrà visualizzare una serie di MDSLIRow rappresentanti tutte le DSLI aziendali.

Relazioni con altre classi

FrontEnd::View::Components
FrontEnd::Actions::rootAction

4.4.4 FrontEnd ::View :: Containers :: UserManagment

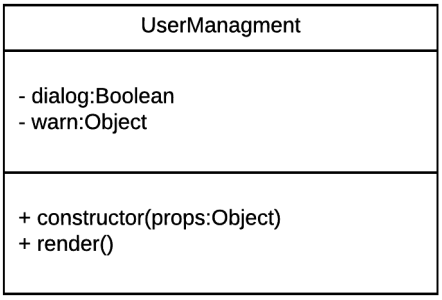

Fig 44: UserManagment

Descrizione

Container che rappresenta la pagina web per la gestione degli utenti.

Utilizzo

La pagina dovrà visualizzare una serie di MUserRow rappresentanti tutti gli utenti appartenenti all'istanza aziendale. Sarà presente inoltre una MTextBox e una comboBox per invitarne di nuovi e una dialog per confermare eventuali eliminazioni.

Relazioni con altre classi

FrontEnd::View::Components
FrontEnd::Actions::rootAction

4.4.5 FrontEnd ::View :: Containers :: Editor

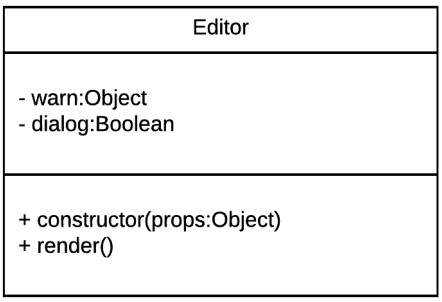

Fig 45: Editor

Descrizione

Container che rappresenta la pagina web per la modifica di DSLI.

Utilizzo

La pagina dovrà visualizzare il nome della DSLI e il relativo codice in una MTextBox. Sarà presente un pulsante per attivare la dialog di rinominazione della DSLI e uno per salvare le modifiche effettuate.

Attributi

- *-warn:Object*: Messaggio di avvertimento per l'utente in caso i dati non siano validi;
- *-dialog:Boolean*: Valore booleano che indica se la dialog debba essere visibile.

Relazioni con altre classi

FrontEnd::View::Components
FrontEnd::Actions::rootAction

4.4.6 FrontEnd ::View :: Containers :: Header

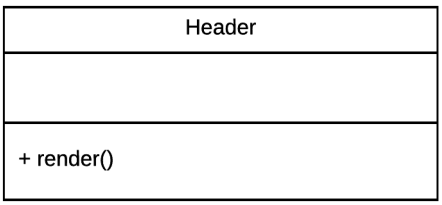

Fig 46: Header

Descrizione

Container che rappresenta l'intestazione della single page application.

Utilizzo

La banda superiore dovrà visualizzare il logo di MaaS oltre all'insieme di macro-funzionalità offerte dal sistema. Ognuna di queste sarà un link per la pagina corrispondente.

Relazioni con altre classi

FrontEnd :: View :: Containers :: MainPage
FrontEnd::View::Components
FrontEnd::Actions::rootAction

4.4.7 FrontEnd ::View :: Containers :: HomeDeveloper

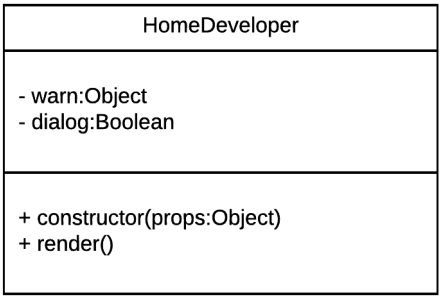

Fig 47: HomeDeveloper

Descrizione

Container che rappresenta la pagina principale dei super-admin.

Utilizzo

La pagina dovrà visualizzare pulsante per la impersonificazione di un utente, e il corrispettivo modal.

Relazioni con altre classi

FrontEnd::View::Components
FrontEnd::Actions::rootAction

4.4.8 FrontEnd ::View :: Containers :: HomePage

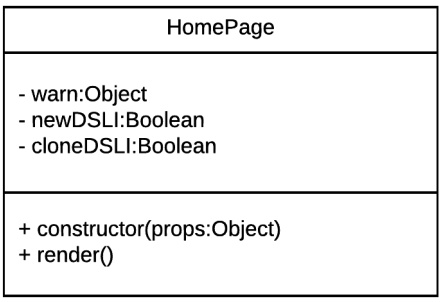

Fig 48: HomePage

Descrizione

Container che rappresenta la pagina principale degli utenti autenticati.

Utilizzo

La pagina dovrà visualizzare una serie di MDSLIRow rappresentanti tutte le DSLI a cui l'utente ha accesso. Deve inoltre essere presente un pulsante per la creazione di una nuova DSLI.

Relazioni con altre classi

FrontEnd::View::Components
FrontEnd::Actions::rootAction

4.4.9 FrontEnd ::View :: Containers :: Login

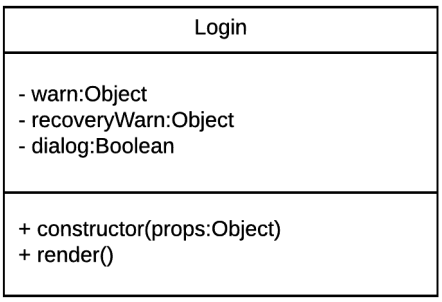

Fig 49: Login

Descrizione

Container che rappresenta la pagina web per l'autenticazione utente.

Utilizzo

La pagina dovrà visualizzare due MTextBox, una per l'email e una per la password. L'apposita action deve poi completare la procedura attraverso le API.

Attributi

- *-warn:Object*: Messaggio di avvertimento per l'utente in caso i dati non siano validi.

Relazioni con altre classi

FrontEnd::View::Components
FrontEnd::Actions::rootAction

4.4.10 FrontEnd ::View :: Containers :: MainPage

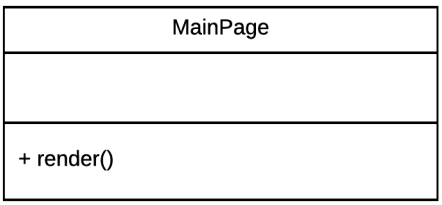

Fig 50: MainPage

Descrizione

Container che rappresenta la pagina principale degli utenti non autenticati.

Utilizzo

La pagina dovrà presentare MaaS al pubblico, elencandone i pregi.

4.4.11 FrontEnd ::View :: Containers :: Profile

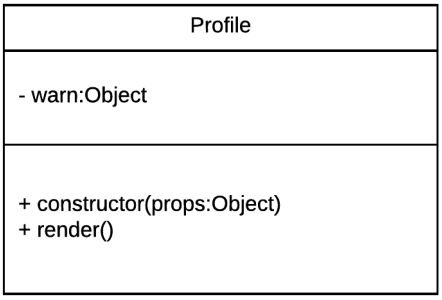

Fig 51: Profile

Descrizione

Container che rappresenta la pagina web per la visualizzazione del proprio account.

Utilizzo

La pagina dovrà visualizzare i dati del proprio account e un collegamento per il cambio di password.

Relazioni con altre classi

FrontEnd::View::Components
FrontEnd::Actions::rootAction

4.4.12 FrontEnd ::View :: Containers :: Provider

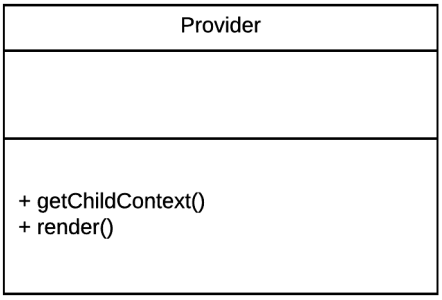

Fig 52: Provider

Descrizione

Container vuoto in cima alla gerarchia.

Utilizzo

Questo container è necessario per mantenere un contesto accessibile a qualsiasi componente figlio.

4.4.13 FrontEnd ::View :: Containers :: RecoverPassword

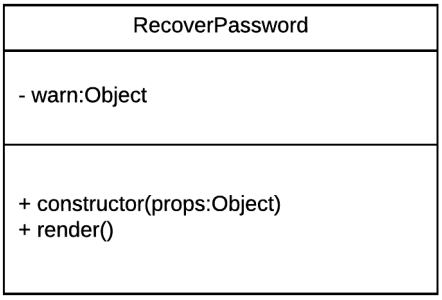

Fig 53: RecoverPassword

Descrizione

Container che rappresenta la pagina web per la seconda parte della procedura di recupero password.

Utilizzo

La pagina dovrà visualizzare due MTextBox per inserire la nuova password e controllare di averla digitata correttamente.

Relazioni con altre classi

FrontEnd::View::Components
FrontEnd::Actions::rootAction

4.4.14 FrontEnd ::View :: Containers :: SignUp

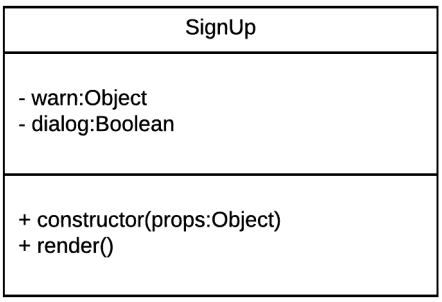

Fig 54: SignUp

Descrizione

Container che rappresenta la pagina web per registrazione di un'istanza aziendale.

Utilizzo

La pagina dovrà visualizzare due MTextBox per inserire la propria mail e il nome dell'azienda da registrare.

Attributi

- *-warn:Object*: Messaggio di avvertimento per l'utente in caso i dati non siano validi.

Relazioni con altre classi

FrontEnd::View::Components
FrontEnd::Actions::rootAction

4.5 Model

4.5.1 FrontEnd :: View :: Model :: CellModel

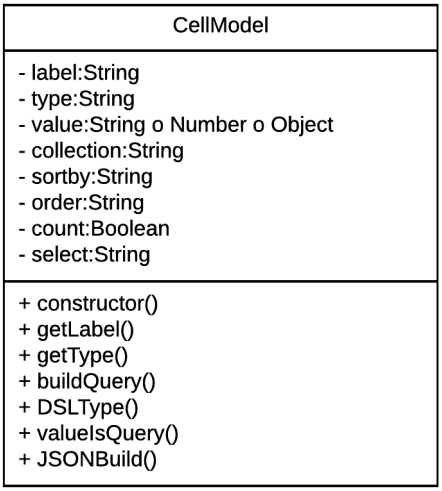

Fig 55: CellModel

Descrizione

Classe che definisce il modello di una DSL di tipo cell.

Utilizzo

L'oggetto che viene creato rappresenta la DSL ti tipo cell. L'oggetto fornisce i dati presenti nella DSL e crea la query che andrà ad essere eseguita nel server.

Attributi

- *-label:String*: Stringa che rappresenta il nome della cella che verrà eseguita;
- *-type:String*: Stringa che indica come viene rappresentato il valore della cella che verrà eseguita;
- *-value:String o Number o Object*: L'attributo rappresenta i dati da visualizzare durante l'esecuzione se è di tipo String o Number oppure contiene i dati per costruire una query da inviare al server;
- *-collection:String*: Rappresenta una collection del server che dovrà contenere il dato cercato;
- *-sortby:String*: mRapresenta l'atributo con cui si ordina la collection;
- *-order:String*: Rappresenta come ordinare la collection, in ordine crescente o decrescente;
- *-count:Boolean*: Se true l'esecuzione della query ritorna un numero, che rappresenta il numero delle righe che la query che ha selezionato;

- *-select:String*: Seleziona l'attributo presente nel documento di ritorno dopo l'esecuzione della query.

Metodi

- *+constructor()*: Metodo costruttore della classe. Prende in input il parametro params contente la label, il type e la value della cell. Attraverso AttributeReader sarà possibile estrapolare le variabili necessarie;
- *+getLabel()*: Ritorna l'attributo label di cell;
- *+getType()*: Ritorna l'attributo type di cell;
- *+buildQuery()*: Costruisce la query da lanciare al database. se il tipo di value è object altrimenti se il tipo è string o number ritorna il valore di value;
- *+DSLType()*: Ritorna una stringa che indica il tipo di DSL;
- *+valueIsQuery()*: Ritorna un booleano. Se il booleano è true vuol dire che l'attributo value è un oggetto contenente una query. Se il booleano ritorna false allora l'attributo value è una stringa oppure un nuber;
- *+JSONBuild()*: Costruisce il JSON contente i dati restituiti dalla query e i dati che servono a far visualizzare correttamente i dati restituiti.

Relazioni con altre classi

FrontEnd::View::Utils::AttributeReader

4.5.2 FrontEnd :: View :: Model :: CollectionModel

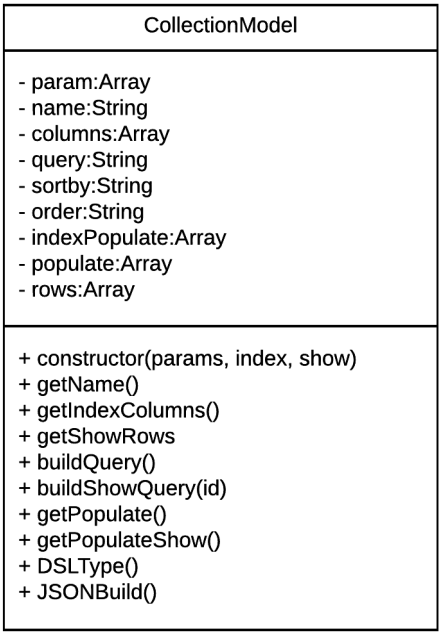

Fig 56: CollectionModel

Descrizione

Classe che definisce il modello di una DSL di tipo collection.

Utilizzo

L'oggetto che viene creato rappresenta la DSL di tipo collection. Fornisce i dati presenti nella DSL e crea la query che andrà al server.

Attributi

- *-param:Array*: Contiene gli elementi della collection e quelli dell'indexModel;
- *-name:String*:Contiene il nome della collection che varrà utilizzata per costruire la query;
- *-columns:Array*: Rappresenta gli attributi del index che verranno visualizzati attraverso le colonne;
- *-query:String*: Rappresenta la query da eseguire da eseguire sulla collection selezionata;
- *-sortby:String*: Rappresenta come ordinare la collection selezionata attraverso un attributo;
- *-order:String*: Rappresenta come ordinare la collection selezionata attraverso un attributo;
- *-indexPopulate:Array*: L'array contiene degli oggetti che servono per effettuare le join tra index e altri documenti;

- *-populate:Array*: L'array contiene degli oggetti che servono per effettuare le join tra show e altri documenti;
- *-rows:Array*: Rappresenta gli attributi del index che verranno visualizzati attraverso le righe.

Metodi

- *+constructor(params, index, show)*: Metodo costruttore della classe. Prende in input il parametro params contente la gli attributi che definisco la collection, index che contiene gli attributi della query da mandare al server e il parametro show che contiene gli attributi che servono a far visualizzare una riga in dettaglio;
- *+getName()*: Ritorna il nome della collection da cercare;
- *+getIndexColumns()*: Ritorna le colonne dell'indexModel;
- *+getShowRows()*: Ritorna le righe dello showModel;
- *+buildQuery()*: Restituisce una query che ritorna la collection;
- *+buildShowQuery(id)*: Restituisce una query che va a la ricerca del documento da visualizzare in base all'id;
- *+getPopulate()*: Ritorna il populate di index;
- *+getPopulateShow()*: Ritorna il populate di show;
- *+DSLType()*: Ritorna una stringa che indica il tipo di DSL;
- *+JSONbuild()*: Costruisce il JSON contente i dati restituiti dalla query e i dati che servono a far visualizzare correttamente i dati restituiti.

Relazioni con altre classi

FrontEnd::View::Utils::AttributeReader

4.5.3 FrontEnd :: View :: Model :: DashboardModel

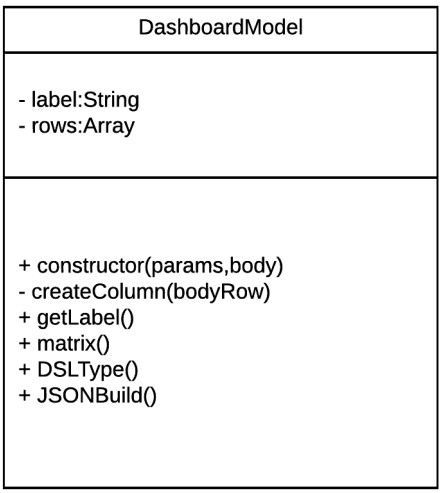

Fig 57: DashboardModel

Descrizione

Classe che definisce il modello di una DSL di tipo dashboard.

Utilizzo

L'oggetto che viene creato rappresenta la DSL di tipo dashboard. Fornisce i dati presenti nella DSL e crea le query che andranno ad interrogare il server di MaaS per eseguire le DSL presenti all'interno della dashboard.

Attributi

- *-label:String*: Rappresenta la label che verrà visualizzata nella dashboard;
- *-rows:Array*: Rappresenta una matrice in cui le righe contengono le colonne. Ogni colonna contiene un oggetto con i suoi dati.

Metodi

- *+constructor(params,body)*: Metodo costruttore della classe. Prende in input il parametro params ,che contiene l'attributo name, e l'attributo body che contiene le righe che vengo inserite in rows;
- *-createColumn(bodyRow)*: Metodo chiamato dal costruttore che serve a costruire le righe partendo dai dati passati per parametro;
- *+getLabel()*: Ritorna l'attributo label;
- *+matrix()*: Ritorna un oggetto contente due array una rappresenta la label delle dsl. L'atro ne contiene i dati;
- *+JSONbuild()*: Costruisce il JSON contente i dati restituiti dalla query e i dati che servono a far visualizzare correttamente i dati restituiti;

- *+DSLType()*: Ritorna una stringa che indica il tipo di DSL.

Relazioni con altre classi

FrontEnd::View::Utils::AttributeReader

4.5.4 FrontEnd :: View :: Model :: DocumentModel

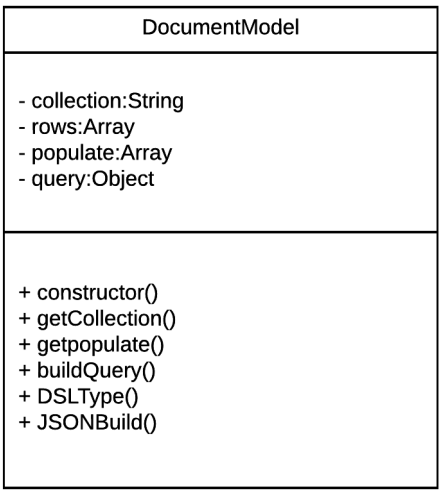

Fig 58: DocumentModel

Descrizione

Classe che definisce il modello di una DSL di tipo document.

Utilizzo

L'oggetto che viene creato rappresenta la DSL di tipo document. Fornisce i dati presenti nella DSL e crea le query che andranno ad interrogare il server.

Attributi

- *-collection:String*:Rappresenta la collection presente nel server che dovrà contenere il dato cercato;
- *-name:String* Rappresenta il nome del document presente nella collection;
- *-rows:Array*:Rappresenta le righe del document che saranno visualizzate;
- *-populate:Array*:Rappresenta un array di oggetti che servono a popolare la lista di document;
- *-query:Object*:Rappresenta l'attributo che servirà a costruire la query che verrà lanciata nel server.

Metodi

- *+constuctor(params, populate, bodyRows)*: Costruttore di DocumentModel, riceve in input i parametri del document, la populate delle righe e le righe da visualizzare;
- *-buildQuery()*: Ritorna la query che verrà eseguita nel server;
- *+getCollection()*: Ritorna la collection che viene eseguita nella query;

- *+getPopulate()*:Ritorna l'attributo populate;
- *+JSONbuild()*: Costruisce il JSON contente i dati restituiti dalla query e i dati che servono a far visualizzare correttamente i dati restituiti;
- *+DSLType()*: Ritorna una stringa che indica il tipo di DSL.

Relazioni con altre classi

FrontEnd::View::Utils::AttributeReader

4.6 Utils

4.6.1 FrontEnd :: Utils :: SyntaxChecker

Descrizione

Classe dedicata al controllo sintattico delle DSLI.

Utilizzo

Grazie ai suoi metodi si sarà in grado di poter segnalare all'utente se una DSLI è sintatticamente corretta.

4.6.2 FrontEnd :: Utils :: PageBuilder

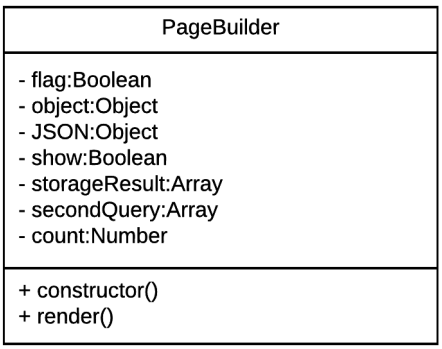

Fig 59: PageBuilder

Descrizione

Classe dedicata alla compilazione delle DSL e alla realizzazione di una pagina web per visualizzare i risultati di una query.

Utilizzo

Nel costruttore avviene la compilazione e l'esecuzione della DSL. Interpretando i dati ricevuti, questa classe crea un Component per la loro visualizzazione.

Attributi

- *-flag:Boolean*: Booleano che indica se è stata eseguita oppure no l'invio della query al server;
- *-object:Object*: Oggetto che Ritorna dopo l'esecuzione della DSL. Può essere di tipo cell, document, dashboard oppure collection;
- *-JSON:JSON*:Ritorna un JSON contente gli attributi e le label necessarie alla costruzione visuale della query eseguita;
- *-show:Boolean*: booleano che indica se si può far partire il render della pagina;
- *-storageResult:Array*: Array che contiene il ritorno della query passata al server;
- *-secondQuery:Array*: Array di Array che contiene i dati ritornati di tutte le query che servono alla popolazione di storageResult;
- *-count:Number*: contatore di secondQuery.

Metodi

- *+constructor()*: Costruttore di pagebuilder compila ed esegue il compilato che ritorna un oggetto chiamato object;
- *+Render()*: Visualizza i dati restituiti dalla query eseguita dal server .

Relazioni con altre classi

FrontEnd::View::Utils::AttributeReader
FrontEnd :: View :: Model :: CellModel
FrontEnd :: View :: Model :: CollectionModel
FrontEnd :: View :: Model :: DashboardModel
FrontEnd :: View :: Model :: DocumentModel

4.6.3 FrontEnd :: Utils :: AttributeReader

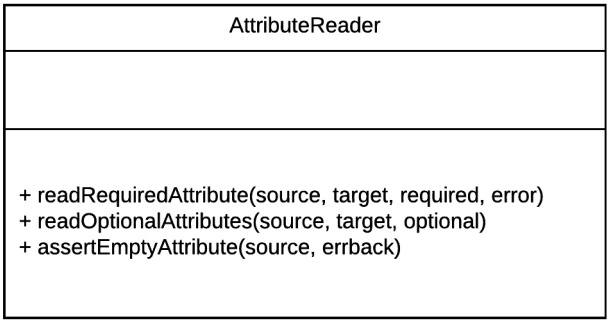

Fig 60: AttributeReader

Descrizione

Classe, presa da MaaP, che ha lo scopo di leggere gli attributi sottoforma di oggetti presenti in altre classi e d'assegnarli a queste ultime.

Utilizzo

I metodi della classe ricercano degli attributi necessari ed opzionali nei propri parametri.

Metodi

- *+readRequiredAttributes(source, target, required, errback)*: Metodo che cerca i campi in un Oggetto e se li trova li assegna alla classe chiamante. Se non trova un parametro richiesto restituisce un errore;
- *+readOptionalAttributes(source,target,optional)*: Metodo che cerca i campi opzionali in un Oggetto e se li trova li assegna alla classe chiamante;
- *+assertEmptyAttributed(source,errback)*: Controlla che i campi del Oggetto non siano vuoti.

4.6.4 FrontEnd :: Utils :: MaaSError

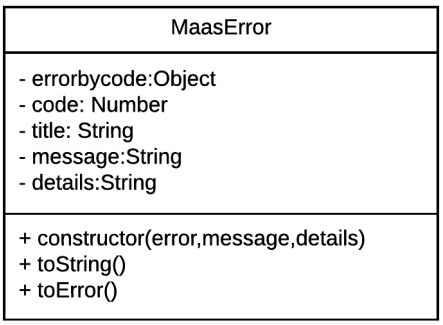

Fig 61: MaaSError

Descrizione

Classe, che serve a raccogliere gli errori e segnalarli.

Utilizzo

I metodi raccoglieranno gli errori e provvederanno a segnalarli.

Attributi

- *-errorbycode:Object*:oggetto che contiene tutti gli identificativi degli errori;
- *-code:Number*:Rappresenta il codice dell'errore;
- *-title:String*:Rappresenta il titolo dell'errore;
- *-mesagge:String*:Rappresenta il messaggio dell'errore;
- *-datails:String*:Rappresenta i dettagli dell'errore.

Metodi

- *+constructor(error,mesagge,details)*: Metodo che viene chiamato durante il lancio di un errore;
- *+toString()*: Ritorna l'errore sottoforma di stringa;
- *+toError()*: Ritorna l'errore sottoforma di un nuovo errore.

5 Diagrammi di Sequenza

5.1 Ciclo di vita di una Action

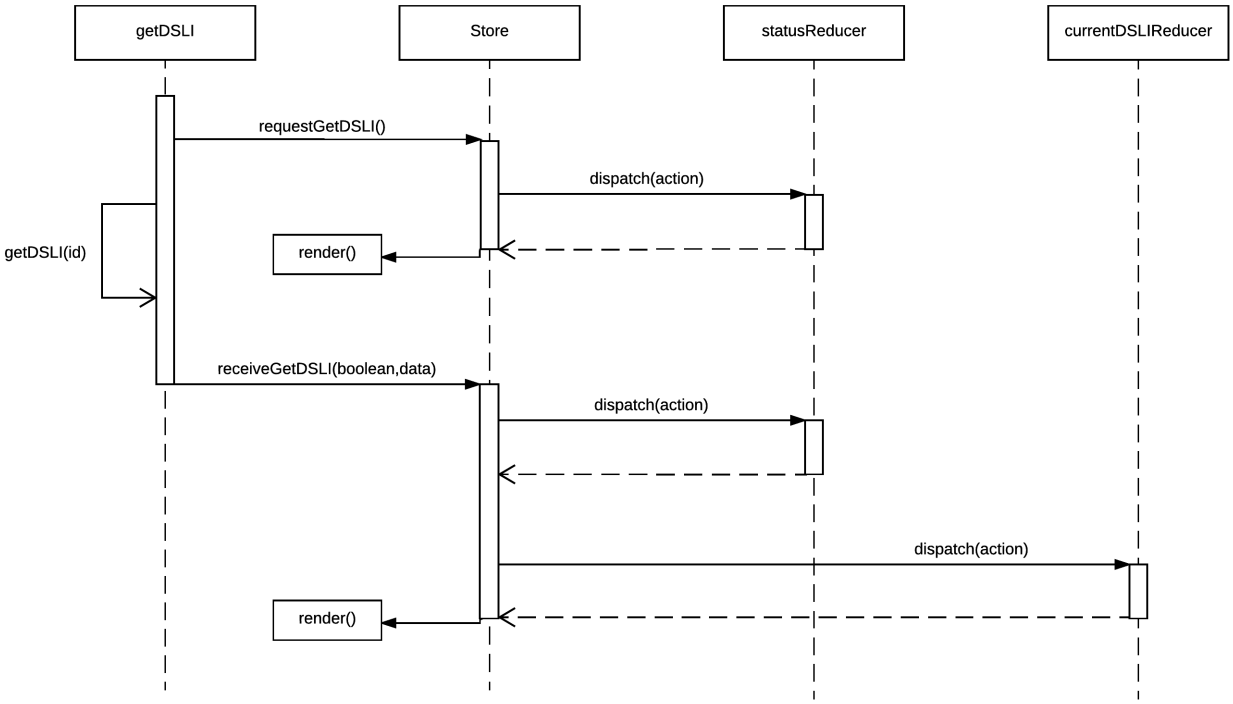

Fig 62: Action

Il diagramma sopra riportato mostra il ciclo di vita di una action dal suo lancio alla sua terminata esecuzione.

5.2 Ciclo di vita di una DSLI

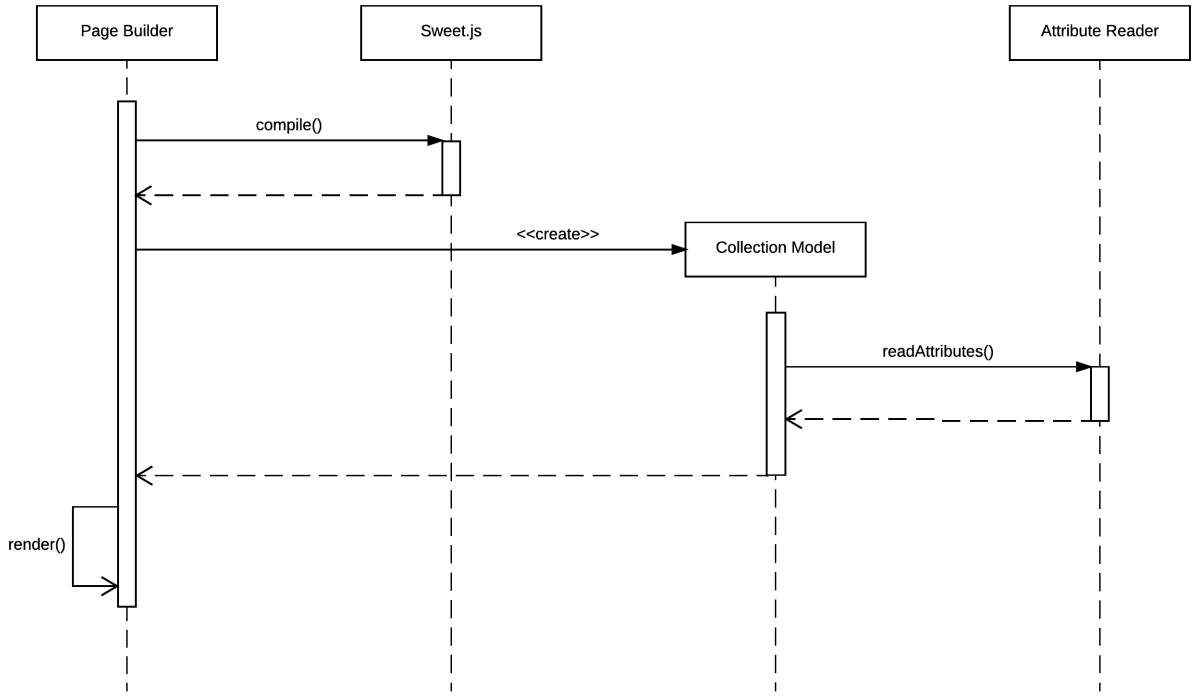

Fig 63: Engine

Il diagramma sopra riportato mostra il ciclo di vita di una DSLI da quando viene ricevuto il testo fino alla visualizzazione dei dati.