



# Definizione di Prodotto

*Gruppo MINT – Progetto MaaS*

## Informazioni sul documento

<b>Versione</b>	2.0.0
<b>Redazione</b>	Navid Taha, Fabiano Tavallini, Tommaso Zagni
<b>Verifica</b>	Michael Ogbuachi, Thomas Fuser
<b>Approvazione</b>	Enrico Canova
<b>Uso</b>	Esterno
<b>Distribuzione</b>	Prof. Tullio Vardanega Prof. Riccardo Cardin Gruppo MINT

## Descrizione

Questo documento descrive l'architettura del prodotto e la progettazione di dettaglio adottata nella realizzazione del progetto MaaS.

## Registro delle modifiche

Versione	Data	Collaboratori	Descrizione
2.0.0	19-07-2016	Michael Ogbuachi (Responsabile)	Approvazione del documento.
1.1.0	18-07-2016	Fabiano Tavallini, Enrico Canova (Verificatori)	Verifica del documento.
1.0.3	16-07-2016	Thomas Fuser, Navid Taha (Progettisti)	I diagrammi di attività sono stati rimossi dal documento in quanto ritenuti non idonei ad esso.
1.0.2	15-07-2016	Thomas Fuser, Tommaso Zagni (Progettisti)	Ampliate le descrizioni di alcuni dei metodi più significativi del Back-end. Rivisitata la sezione "Tecnologie utilizzate", discutendone in modo più equilibrato i vantaggi e gli svantaggi. Agg
1.0.1	14-07-2016	Michael Ogbuachi, Navid Taha (Progettisti)	Aggiunta la sezione "Oggetti JSON" per chiarire gli input e gli output di tipo JSON.
1.0.0	07-06-2016	Enrico Canova (Responsabile)	Approvazione del documento.
0.4.0	06-06-2016	Michael Ogbuachi (Verificatore)	Verifica del documento.
0.3.0	05-06-2016	Thomas Fuser (Verificatore)	Verifica del documento.
0.2.6	04-06-2016	Michael Ogbuachi (Progettista)	Sezione 9: stesura Tracciamento metodi-test.
0.2.5	02-06-2016	Navid Taha, Thomas Fuser (Progettisti)	Sezione 6: fine stesura metodi del Front-end.
0.2.4	31-05-2016	Fabiano Tavallini, Tommaso Zagni (Progettisti)	Sezione 5: fine stesura metodi del Back-end.
0.2.3	20-05-2016	Navid Taha, Thomas Fuser (Progettisti)	Sezione 6: inizio stesura metodi delle classi del Front-end.
0.2.2	18-05-2016	Fabiano Tavallini, Tommaso Zagni (Progettisti)	Sezione 5: inizio stesura metodi delle classi del Back-end.
0.2.1	16-05-2016	Fabiano Tavallini, Tommaso Zagni (Progettisti)	Sezione 7: inizio stesura richieste HTTP.
0.2.0	16-05-2016	Michael Ogbuachi (Verificatore)	Verifica del documento.
0.1.0	15-05-2016	Thomas Fuser (Verificatore)	Verifica del documento.

0.0.16	14-05-2016	Michael Ogbuachi (Progettista)	Sezione 9: stesura Tracciamento requisiti-classi.
0.0.15	13-05-2016	Michael Ogbuachi (Progettista)	Sezione 9: stesura Tracciamento classi-requisiti.
0.0.14	13-05-2016	Thomas Fuser (Progettista)	Completata la progettazione delle classi del Front-End.
0.0.13	12-05-2016	Enrico Canova (Progettista)	Completata la progettazione delle classi package del Back-End.
0.0.12	12-05-2016	Michael Ogbuachi (Progettista)	Sezione 9: stesura Tracciamento requisiti-componenti.
0.0.11	11-05-2016	Michael Ogbuachi (Progettista)	Sezione 9: stesura Tracciamento componenti-requisiti.
0.0.10	10-05-2016	Thomas Fuser (Progettista)	Completati i package del Front-End.
0.0.9	09-05-2016	Enrico Canova (Progettista)	Completati i package del Back-End.
0.0.8	07-05-2016	Michael Ogbuachi, Thomas Fuser (Progettisti)	Appendici Pattern.
0.0.7	06-05-2016	Tommaso Zagni (Progettista)	Descrizione View.
0.0.6	05-05-2016	Tommaso Zagni (Progettista)	Inizio progettazione Front-End.
0.0.5	05-05-2016	Fabiano Tavallini (Progettista)	Inizio progettazione Back-End.
0.0.4	30-04-2016	Tommaso Zagni (Progettista)	Inizio Descrizione architetturale.
0.0.3	29-04-2016	Thomas Fuser, Enrico Canova (Progettisti)	Aggiunta sezione tecnologie utilizzate.
0.0.2	27-04-2016	Thomas Fuser, Enrico Canova (Progettisti)	Sezione 1: inizio stesura Introduzione.
0.0.1	25-04-2016	Navid Taha (Progettista)	Definizione scheletro del documento.

## Indice

<b>1 Introduzione</b>	<b>4</b>
1.1 Scopo del documento . . . . .	4
1.2 Scopo del prodotto . . . . .	4
1.3 Glossario . . . . .	4
1.4 Riferimenti . . . . .	4
1.4.1 Normativi . . . . .	4
1.4.2 Informativi . . . . .	4
<b>2 Standard di progetto</b>	<b>6</b>
2.1 Standard di progettazione architetturale . . . . .	6
2.2 Standard di documentazione del codice . . . . .	6
2.3 Standard di denominazione di entità e relazioni . . . . .	6
2.4 Standard di programmazione . . . . .	6
2.5 Strumenti di lavoro . . . . .	6
<b>3 Tecnologie utilizzate</b>	<b>7</b>
3.1 Node.js . . . . .	7
3.1.1 Vantaggi . . . . .	7
3.1.2 Svantaggi . . . . .	7
3.2 MongoDB . . . . .	7
3.2.1 Vantaggi . . . . .	7
3.2.2 Svantaggi . . . . .	8
3.3 React . . . . .	8
3.3.1 Vantaggi . . . . .	8
3.3.2 Svantaggi . . . . .	8
3.4 LoopBack . . . . .	9
3.4.1 Vantaggi . . . . .	9
3.4.2 Svantaggi . . . . .	9
3.5 HTML5 . . . . .	9
3.5.1 Vantaggi . . . . .	9
3.5.2 Svantaggi . . . . .	10
3.6 CSS . . . . .	10
3.6.1 Vantaggi . . . . .	10
3.6.2 Svantaggi . . . . .	10
3.7 Sweet.js . . . . .	10
3.7.1 Vantaggi . . . . .	10
3.7.2 Svantaggi . . . . .	10
<b>4 Descrizione architetturale</b>	<b>11</b>
4.1 Metodo e formalismo di specifica . . . . .	11
4.2 Architettura generale . . . . .	12
4.2.1 Back-end . . . . .	13
4.2.2 Front-end . . . . .	14
4.3 Interfaccia REST . . . . .	15
<b>5 Back-end</b>	<b>17</b>
5.1 Descrizione packages e classi . . . . .	17

5.1.1	Back-end . . . . .	17
5.1.1.1	Informazioni sul package . . . . .	18
5.1.1.2	Classi . . . . .	19
5.1.2	Back-end::Models . . . . .	21
5.1.2.1	Informazioni sul package . . . . .	21
5.1.2.2	Classi . . . . .	22
5.1.3	Back-end::Models::DSLISModels . . . . .	33
5.1.3.1	Informazioni sul package . . . . .	33
5.1.3.2	Classi . . . . .	33
5.1.4	Back-end::Datasources . . . . .	54
5.1.4.1	Informazioni sul package . . . . .	54
5.1.4.2	Classi . . . . .	55
5.1.5	Back-end::Connectors . . . . .	56
5.1.5.1	Informazioni sul package . . . . .	56
5.1.5.2	Classi . . . . .	57
5.1.6	Back-end::Middlewares . . . . .	58
5.1.6.1	Informazioni sul package . . . . .	58
5.1.6.2	Classi . . . . .	58
5.2	Interfaccia REST . . . . .	63
5.2.1	Comunicazione tra client e server . . . . .	63
5.2.1.1	Oggetti JSON . . . . .	63
5.2.2	Permessi . . . . .	65
5.2.3	Richieste HTTP . . . . .	66
5.2.3.1	Richieste Users . . . . .	66
5.2.3.2	Richieste Super Admin . . . . .	76
5.2.3.3	Richieste Companies . . . . .	79
5.2.3.4	Richieste Databases . . . . .	85
5.2.3.5	Richieste Collections . . . . .	91
5.2.3.6	Richieste Documents . . . . .	102
5.2.3.7	Richieste Dashboards . . . . .	114
5.2.3.8	Richieste Cell . . . . .	123
<b>6</b>	<b>Front-end</b> . . . . .	<b>133</b>
6.1	Descrizione packages e classi . . . . .	133
6.1.1	Front-end . . . . .	133
6.1.1.1	Informazioni sul package . . . . .	134
6.1.1.2	Classi . . . . .	135
6.1.2	Front-end::ActionCreators . . . . .	143
6.1.2.1	Informazioni sul package . . . . .	143
6.1.2.2	Classi . . . . .	144
6.1.3	Front-end::ActionCreators::DSLISActionCreators . . . . .	160
6.1.3.1	Informazioni sul package . . . . .	160
6.1.3.2	Classi . . . . .	160
6.1.4	Front-end::WebAPIUtils . . . . .	181
6.1.4.1	Informazioni sul package . . . . .	181
6.1.4.2	Classi . . . . .	182
6.1.5	Front-end::WebAPIUtils::DSLISWebAPIUtils . . . . .	194
6.1.5.1	Informazioni sul package . . . . .	194
6.1.5.2	Classi . . . . .	195

6.1.6	Front-end::Stores . . . . .	209
6.1.6.1	Informazioni sul package . . . . .	209
6.1.6.2	Classi . . . . .	210
6.1.7	Front-end::Stores::DSLISStores . . . . .	218
6.1.7.1	Informazioni sul package . . . . .	218
6.1.7.2	Classi . . . . .	218
6.1.8	Front-end::Views . . . . .	225
6.1.8.1	Informazioni sul package . . . . .	225
6.1.8.2	Classi . . . . .	226
<b>7</b>	<b>Design pattern</b>	<b>289</b>
7.1	Design Pattern Architetturali . . . . .	289
7.1.1	Flux . . . . .	289
7.1.1.1	Scopo . . . . .	289
7.1.1.2	Utilizzo . . . . .	289
7.1.2	Constructor injection . . . . .	291
7.1.2.1	Scopo . . . . .	291
7.1.2.2	Utilizzo . . . . .	291
7.2	Design Pattern Strutturali . . . . .	291
7.2.1	Module . . . . .	291
7.2.1.1	Scopo . . . . .	291
7.2.1.2	Utilizzo . . . . .	291
7.3	Design Pattern Comportamentali . . . . .	292
7.3.1	Chain of Responsibility . . . . .	292
7.3.1.1	Scopo . . . . .	292
7.3.1.2	Utilizzo . . . . .	292
7.3.2	Strategy . . . . .	293
7.3.2.1	Scopo . . . . .	293
7.3.2.2	Utilizzo . . . . .	293
7.3.3	Observer . . . . .	294
7.3.3.1	Scopo . . . . .	294
7.3.3.2	Utilizzo . . . . .	294
<b>8</b>	<b>Tracciamento</b>	<b>296</b>
8.1	Tracciamento componenti-requisiti . . . . .	296
8.2	Tracciamento requisiti-componenti . . . . .	302
8.3	Tracciamento classi-requisiti . . . . .	314
8.4	Tracciamento requisiti-classi . . . . .	328
8.5	Tracciamento metodi-test . . . . .	344
<b>Appendici</b>		<b>348</b>
<b>A</b>	<b>Descrizione Design Pattern</b>	<b>348</b>
A.1	Design Pattern Architetturali . . . . .	348
A.1.1	Flux . . . . .	348
A.1.2	Constructor Injection . . . . .	349
A.2	Design Pattern Strutturali . . . . .	349
A.2.1	Module . . . . .	349
A.3	Design Pattern Comportamentali . . . . .	350
A.3.1	Observer . . . . .	350

A.3.2	Strategy . . . . .	350
A.3.3	Chain of Responsibility . . . . .	351

## Elenco delle figure

1	Diagramma del Deployment . . . . .	12
2	Architettura di un'applicazione LoopBack . . . . .	13
3	Diagramma dei package del back-end . . . . .	14
4	Architettura Flux . . . . .	14
5	Diagramma dei package del back-end . . . . .	17
6	Diagramma delle classi del back-end . . . . .	18
7	Diagramma della classe Application . . . . .	19
8	Diagramma della classe Bootstrapper . . . . .	19
9	Diagramma delle classi del package Models . . . . .	21
10	Diagramma della classe UserModel . . . . .	22
11	Diagramma della classe SuperAmministratoreModel . . . . .	26
12	Diagramma della classe CompanyModel . . . . .	28
13	Diagramma della classe ExternalDatabaseModel . . . . .	30
14	Diagramma delle figure del package: Back-end::Models::DSLISModels . . . . .	33
15	Diagramma della classe DSLISModel . . . . .	34
16	Diagramma della classe ConvertTo . . . . .	35
17	Diagramma della classe ConvertToImp . . . . .	36
18	Diagramma della classe CollectionModel . . . . .	37
19	Diagramma della classe DocumentModel . . . . .	41
20	Diagramma della classe DashboardModel . . . . .	46
21	Diagramma della classe CellModel . . . . .	49
22	Diagramma della classe DSLInterpreterStategy . . . . .	52
23	Diagramma della classe DSLInterpreter . . . . .	53
24	Diagramma delle classi del package Datasources . . . . .	54
25	Diagramma delle classi del package Connectors . . . . .	56
26	Diagramma delle classi del package Middlewares . . . . .	58
27	Diagramma della classe MiddlewareHandler . . . . .	59
28	Diagramma della classe UrlNotFoundHandler . . . . .	60
29	Diagramma della classe ServerErrorHandler . . . . .	61
30	Diagramma della classe RequestHandler . . . . .	62
31	Diagramma di sequenza: POST /User . . . . .	66
32	Diagramma di sequenza: POST /Users/login . . . . .	67
33	Diagramma di sequenza: PUT /Users . . . . .	68
34	Diagramma di sequenza: POST /Users/logout . . . . .	69
35	Diagramma di sequenza: GET /Users . . . . .	70
36	Diagramma di sequenza: GET /Users/{email} . . . . .	71
37	Diagramma di sequenza: DELETE /Users/deleteUser/{email} . . . . .	72
38	Diagramma di sequenza: GET /User/SearchUser/{value} . . . . .	73
39	Diagramma di sequenza: POST /Users/forgotPassword . . . . .	74
40	Diagramma di sequenza: PUT /Users/changeRole . . . . .	75
41	Diagramma di sequenza: POST /Users/sendInvite . . . . .	76
42	Diagramma di sequenza: POST /SuperAdmin/login . . . . .	77
43	Diagramma di sequenza: POST /SuperAdmin/logout . . . . .	78

44	Diagramma di sequenza: POST /SuperAdmin/impersonateUser . . . . .	79
45	Diagramma di sequenza: GET /Companies . . . . .	80
46	Diagramma di sequenza: POST /Companies . . . . .	81
47	Diagramma di sequenza: GET /Companies/{companyName} . . . . .	82
48	Diagramma di sequenza: PUT /Companies . . . . .	83
49	Diagramma di sequenza: DELETE /Companies/deleteCompanies . . . . .	84
50	Diagramma di sequenza: GET /Companies/searchCompany/{value} . . . . .	85
51	Diagramma di sequenza: POST /ExternalDatabases . . . . .	86
52	Diagramma di sequenza: DELETE /ExternalDatabases . . . . .	87
53	Diagramma di sequenza: GET /ExternalDatabases/{query} . . . . .	88
54	Diagramma di sequenza: GET /ExternalDatabases/searchExternalDatabase/{value} . . . . .	89
55	Diagramma di sequenza: POST /ExternalDatabases/allowExternalDatabasesAccess . . . . .	90
56	Diagramma di sequenza: POST /ExternalDatabases/denyExternalDatabaseAccess . . . . .	91
57	Diagramma di sequenza: GET /Collections . . . . .	92
58	Diagramma di sequenza: POST /Collections . . . . .	93
59	Diagramma di sequenza: PUT /Collections . . . . .	94
60	Diagramma di sequenza: DELETE /Collections . . . . .	95
61	Diagramma di sequenza: GET /Collections/retrieveCollectionDSLIS/{id} . . . . .	96
62	Diagramma di sequenza: GET /Collections/searchCollections/{value, email} . . . . .	97
63	Diagramma di sequenza: GET /Collections/ExecuteCollection/{id} . . . . .	98
64	Diagramma di sequenza: GET /Collections/SendEmail/{id, format} . . . . .	99
65	Diagramma di sequenza: GET /Collections/Export/{id, format} . . . . .	100
66	Diagramma di sequenza: GET /Collections/exportCollectionDSLIS/{id} . . . . .	101
67	Diagramma di sequenza: POST /Collections/importCollectionDSLIS . . . . .	102
68	Diagramma di sequenza: GET /Documents . . . . .	103
69	Diagramma di sequenza: POST /Documents . . . . .	104
70	Diagramma di sequenza: PUT /Documents . . . . .	105
71	Diagramma di sequenza: DELETE /Documents . . . . .	106
72	Diagramma di sequenza: GET /Documents/searchDocuments/{value, email} . . . . .	107
73	Diagramma di sequenza: GET /Documents/ExecuteDocument/{id} . . . . .	108
74	Diagramma di sequenza: GET /Documents/ExecuteDocument/{Collection}/{id} . . . . .	109
75	Diagramma di sequenza: GET /Documents/SendEmail/{id, format} . . . . .	110
76	Diagramma di sequenza: GET /Documents/Export/{id, format} . . . . .	111
77	Diagramma di sequenza: GET /Documents/retrieveDocumentDSLIS . . . . .	112
78	Diagramma di sequenza: GET /Documents/exportDocumentDSLIS . . . . .	113
79	Diagramma di sequenza: POST /Documents/importDocumentDSLIS . . . . .	114
80	Diagramma di sequenza: GET /Dashboards . . . . .	115
81	Diagramma di sequenza: POST /Dashboards . . . . .	116
82	Diagramma di sequenza: PUT /Dashboards . . . . .	117
83	Diagramma di sequenza: DELETE /Dashboards . . . . .	118
84	Diagramma di sequenza: GET /Dashboards/retrieveDashboardDSLIS/{id} . . . . .	119
85	Diagramma di sequenza: GET /Dashboards/searchDashboards/{value, email} . . . . .	120
86	Diagramma di sequenza: GET /Dashboards/ExecuteDashboard/{id} . . . . .	121
87	Diagramma di sequenza: GET /Dashboards/exportDashboardDSLIS/{id} . . . . .	122
88	Diagramma di sequenza: POST /Dashboards/importDashboardDSLIS . . . . .	123
89	Diagramma di sequenza: GET /Cells . . . . .	124
90	Diagramma di sequenza: POST /Cells . . . . .	125
91	Diagramma di sequenza: PUT /Cells . . . . .	126
92	Diagramma di sequenza: DELETE /Cells . . . . .	127

93	Diagramma di sequenza: GET /Cells/searchCell/{value, email} . . . . .	128
94	Diagramma di sequenza: GET /Cells/ExecuteCell/{id} . . . . .	129
95	Diagramma di sequenza: GET /Cells/GET /Cells/retrieveCellDSLIS{id} . . . . .	130
96	Diagramma di sequenza: GET /Cells/exportCellDSLIS{id} . . . . .	131
97	Diagramma di sequenza: POST /Cells/importCellDSLIS . . . . .	132
98	Diagramma dei package del Front-end . . . . .	133
99	Diagramma delle classi del Front-end . . . . .	134
100	Diagramma della classe Dispatcher . . . . .	135
101	Diagramma della classe Application . . . . .	136
102	Diagramma della classe Router . . . . .	137
103	Diagramma della classe Constants . . . . .	142
104	Diagramma delle classi del package ActionCreators . . . . .	143
105	Diagramma della classe UserActionCreator . . . . .	144
106	Diagramma della classe SessionActionCreator . . . . .	149
107	Diagramma della classe ExternalDatabaseActionCreator . . . . .	152
108	Diagramma della classe SuperAmministratoreActionCreator . . . . .	155
109	Diagramma della classe CompanyActionCreator . . . . .	157
110	Diagramma delle classi del package DSLISActionCreators . . . . .	160
111	Diagramma della classe CollectionActionCreator . . . . .	161
112	Diagramma della classe DocumentActionCreator . . . . .	166
113	Diagramma della classe DashboardActionCreator . . . . .	172
114	Diagramma della classe CellActionCreator . . . . .	177
115	Diagramma delle classi del package WEBAPIUtils . . . . .	181
116	Diagramma della classe UserWebAPIUtils . . . . .	182
117	Diagramma della classe SessionWebAPIUtils . . . . .	186
118	Diagramma della classe ExternalDatabaseWebAPIUtils . . . . .	188
119	Diagramma della classe CompanyWebAPIUtils . . . . .	191
120	Diagramma della classe SuperAmministratoreWebAPIUtils . . . . .	193
121	Diagramma delle classi del package DSLISWebAPIUtils . . . . .	194
122	Diagramma della classe CollectionWebAPIUtils . . . . .	195
123	Diagramma della classe DocumentWebAPIUtils . . . . .	199
124	Diagramma della classe DashboardWebAPIUtils . . . . .	203
125	Diagramma della classe CellWebAPIUtils . . . . .	206
126	Diagramma delle classi del package Stores . . . . .	209
127	Diagramma della classe SubjectStore . . . . .	210
128	Diagramma della classe UserStore . . . . .	211
129	Diagramma della classe SessionStore . . . . .	212
130	Diagramma della classe SuperAmministratoreStore . . . . .	214
131	Diagramma della classe ExternalDatabaseStore . . . . .	215
132	Diagramma della classe CompanyStore . . . . .	216
133	Diagramma delle classi del package DSLISStores . . . . .	218
134	Diagramma della classe CompanyStore . . . . .	218
135	Diagramma della classe DocumentStore . . . . .	220
136	Diagramma della classe DashboardStore . . . . .	221
137	Diagramma della classe CellStore . . . . .	223
138	Diagramma delle classi del package Views . . . . .	225
139	Diagramma della classe ObserverView . . . . .	226
140	Diagramma della classe RegistrazioneView . . . . .	227
141	Diagramma della classe LoginView . . . . .	229

142	Diagramma della classe RecuperoPasswordView . . . . .	231
143	Diagramma della classe DashboardView . . . . .	233
144	Diagramma della classe DSLISListView . . . . .	236
145	Diagramma della classe RegistrazioneCollaboratoreView . . . . .	239
146	Diagramma della classe ConfermaRegistrazioneView . . . . .	241
147	Diagramma della classe ImpostazioniApplicazioneView . . . . .	243
148	Diagramma della classe ModificaDashboardPredefinitaView . . . . .	245
149	Diagramma della classe ModificaDatiAnagraficiView . . . . .	248
150	Diagramma della classe ModificaPasswordView . . . . .	250
151	Diagramma della classe GestioneDatabaseEsterniView . . . . .	252
152	Diagramma della classe ModificaPermessiAccessoDatabaseEsterniView . . . . .	254
153	Diagramma della classe GestioneDSLISView . . . . .	257
154	Diagramma della classe GestionePermessiDSLISView . . . . .	260
155	Diagramma della classe CreazioneDSLISView . . . . .	263
156	Diagramma della classe GestioneDSLISView . . . . .	266
157	Diagramma della classe ModificaDSLISView . . . . .	267
158	Diagramma della classe VisualizzazioneDSLISView . . . . .	270
159	Diagramma della classe CollectionView . . . . .	273
160	Diagramma della classe DocumentView . . . . .	275
161	Diagramma della classe CellView . . . . .	277
162	Diagramma della classe CellView . . . . .	280
163	Diagramma della classe SuperAmministratoreDashboardView . . . . .	282
164	Diagramma della classe GestioneDatiAziendeView . . . . .	284
165	Diagramma della classe GestioneDatiUtenteView . . . . .	286
166	Contestualizzazione del pattern Flux . . . . .	290
167	Contestualizzazione del pattern Constructor Injection . . . . .	291
168	Contestualizzazione del pattern Chain of Responsibility . . . . .	293
169	Contestualizzazione del pattern Strategy . . . . .	294
170	Struttura logica del pattern Flux . . . . .	348
171	Struttura logica del pattern Observer . . . . .	350
172	Struttura logica del pattern Strategy . . . . .	351
173	Struttura logica del pattern Chain of Responsibility . . . . .	352

## 1 Introduzione

### 1.1 Scopo del documento

Il documento ha lo scopo di definire la progettazione ad alto livello e di dettaglio del prodotto software *MaaS*.

Vengono illustrate le tecnologie, le componenti, le classi con i relativi metodi e i diversi design pattern adottati per la realizzazione del prodotto. Inoltre, verrà definito il tracciamento tra le componenti software ed i requisiti.

### 1.2 Scopo del prodotto

Lo scopo del progetto è quello di realizzare un sistema chiamato MaaS, fruibile dal web (e quindi distribuito), rivolto ai cosiddetti business man, persone con ruolo aziendale chiave che solitamente non sono esperte d'informatica, per aiutarli nelle decisioni di natura amministrativa e commerciale, fornendogli una piattaforma di visualizzazione dei dati salvati in un *database<sub>G</sub>* di facile utilizzo, ma allo stesso tempo dalle buone potenzialità.

### 1.3 Glossario

Al fine di evitare ogni ambiguità relativa al linguaggio impiegato nei documenti viene fornito il *Glossario v4.0.0*, contenente la definizione dei termini marcati con una G pedice.

### 1.4 Riferimenti

#### 1.4.1 Normativi

- **Capitolato d'appalto C4: MaaS: MongoDB as an admin Service:**  
<http://www.math.unipd.it/~tullio/IS-1/2015/Progetto/C4.pdf>;
- *Norme di Progetto v4.0.0*;
- *Analisi dei Requisiti v4.0.0*.

#### 1.4.2 Informativi

- **Documentazione Interfaccia REST:**  
[https://it.wikipedia.org/wiki/Representational\\_State\\_Transfer](https://it.wikipedia.org/wiki/Representational_State_Transfer)  
<http://rest.elkstein.org/>
- **Documentazione ufficiale React/Flux:**  
<https://facebook.github.io/flux/docs/overview.html>  
<http://fancypixel.github.io/blog/2015/01/29/react-plus-flux-backed-by-rails-api-part-2/>
- **Documentazione ufficiale LoopBack:**  
<https://docs.strongloop.com/display/public/LB/LoopBack>

- **Design patterns:**

Design Patterns: elementi per il riuso di software ad oggetti - Pearson Education Italia, 2002.

Capitolo 1 Introduction (How Design Patterns Solve Design Problems, How to Select a Design Pattern, How to Use a Design Pattern), Capitolo 5 Behavioral Patterns (Chain of Responsibility, Observer, Strategy)

## 2 Standard di progetto

### 2.1 Standard di progettazione architetturale

All'interno di questo documento sono stati definiti anche gli standard di progettazione. Essendo JavaScript il linguaggio di programmazione principale (in questo progetto), abbiamo deciso di utilizzare il formalismo UML 2.0. Tuttavia è stata applicata una modifica alla notazione, in modo da rappresentare più chiaramente il tipo di dato di una funzione. Tale modifica consiste nell'uso della stringa `function(<parametri>)`, la quale rappresenta quindi il tipo di dato di una funzione che richiede i parametri `<parametri>`.

### 2.2 Standard di documentazione del codice

Il documento *Norme di Progetto v4.0.0* contiene la descrizione degli standard di documentazione del codice.

### 2.3 Standard di denominazione di entità e relazioni

Tutti gli elementi strutturali, quali package, classi, metodi o attributi, devono avere nomi facilmente comprensibili ed autoesplicativi, di una lunghezza tale da non renderne difficile la lettura. Si è deciso di utilizzare sostantivi per gli oggetti e verbi per azioni e metodi. Le abbreviazioni sono ammesse se:

- Non creano ambiguità;
- Permettono un'immediata comprensione;
- Sono ristrette ad uno specifico contesto.

Sempre all'interno del documento *Norme di Progetto v4.0.0* è possibile trovare una chiara presentazione delle regole tipografiche adottate sia in questo che in altri documenti.

### 2.4 Standard di programmazione

Nel documento *Norme di Progetto v4.0.0* si può trovare la definizione degli standard di programmazione.

### 2.5 Strumenti di lavoro

Nel documento *Norme di Progetto v4.0.0* si può trovare anche l'insieme degli strumenti di lavoro utilizzati, correlato ai metodi di utilizzo scelti per ciascuno strumento.

## 3 Tecnologie utilizzate

In questa sezione vengono illustrate le tecnologie adottate per lo sviluppo del progetto. Per ognuna di esse, verrà indicato l'ambito di utilizzo ed i vantaggi che ne derivano.

### 3.1 Node.js

*Node.js*<sub>G</sub> è un *framework*<sub>G</sub> *event-driven*<sub>G</sub> costruito sul motore *JavaScript*<sub>G</sub> V8 di Google Chrome, per piattaforme *UNIX*<sub>G</sub> like. Node.js usa un modello I/O non bloccante e ad eventi, che lo rendono un framework leggero ed efficiente per l'utilizzo server-side.

#### 3.1.1 Vantaggi

- **Approccio asincrono:** grazie alla modalità event-driven node.js evita il classico modello basato su processi o *thread*<sub>G</sub> concorrenti utilizzato dai classici web server. Tale caratteristica garantisce una migliore efficienza in termini di prestazioni, permettendo la gestione di altri processi durante le attese *runtime*<sub>G</sub> in maniera asincrona.
- **Architettura modulare:** node.js si appoggia allo strumento node *package*<sub>G</sub> manager (npm) che permette un'organizzazione semplice del lavoro grazie alla combinazione di librerie con moduli importati. Consentendo allo sviluppatore quanto di contribuire e accedere ai package messi a disposizione dalla community.

#### 3.1.2 Svantaggi

- **Modello di programmazione asincrono:** il programmatore che si approccia per la prima volta a node.js deve adottare uno stile di programmazione asincrono, il quale risulta essere più complicato della programmazione di input e output lineare a blocchi; il codice prodotto risulta essere più disordinato e meno comprensibile e costringe lo sviluppatore a fare affidamento su *callback*<sub>G</sub> nidificate;
- **Single-Threaded:** node.js non supporta la programmazione multi-threaded, mettendo in coda le richieste in arrivo ed eseguendole in sequenza. In conseguenza a ciò, il tempo necessario per rispondere alle richieste dell'utente aumenta.

### 3.2 MongoDB

*MongoDb*<sub>G</sub> è un database *NoSQL*<sub>G</sub> orientato ai documenti, cross-platform e open source; in MongoDB i dati vengono salvati sotto forma di documenti in stile *JSON*<sub>G</sub> con schemi dinamici chiamati *BSON*<sub>G</sub>.

#### 3.2.1 Vantaggi

- **Schemaless:** non esiste uno schema e questo rende MongoDB molto più flessibile rispetto ad un tradizionale database relazionale;

- **Scalabilità orizzontale:** capacità di crescere o diminuire la scala in funzione alla reale necessità e delle disponibilità;
- **Affidabilità:** la caratteristica che rende MongoDB affidabile è la replicazione su server e la facilità con cui si configura, in particolare consente di proteggere i dati nel caso in cui avvengano fallimenti hardware, software o infrastrutturali;
- **Supporto del polimorfismo:** caratteristica che consente di estendere il proprio modello in modo semplice e veloce;
- **Alte performance:** l'assenza di join permette di non rallentare le operazioni di lettura e scrittura; l'indicizzazione inoltre include gli indici di chiave anche sui documenti innestati e sugli array, permettendo così una rapida interrogazione al database.

### 3.2.2 Svantaggi

- **Assenza di Join:** non è possibile creare delle  $join_G$  come nei database relazionali, il che significa che quando si necessita di tale funzionalità bisogna creare più query e fare il join dei dati manualmente con il codice. Quest'aspetto porta a creare codice non flessibile, in quanto le strutture dei dati potrebbero cambiare;
- **Utilizzo della memoria:** MongoDB ha una naturale tendenza ad utilizzare più memoria di quella necessaria, in quanto deve salvare il nome della chiave di ogni documento.

## 3.3 React

$React_G$  è una libreria JavaScript utile alla costruzione di interfacce grafiche e il suo utilizzo è stato richiesto dal proponente. Esso sarà utilizzato per lo sviluppo del front-end, quindi del lato client, poiché fornisce molti strumenti utili alla progettazione, manutenzione e aggiornamento delle  $UI_G$ . In particolare supporta la definizione di modelli  $HTML_G$  sulla base di uno sviluppo a componenti.

### 3.3.1 Vantaggi

- Sviluppo per componenti: ogni componente è in grado di definire le proprie caratteristiche, quindi l'aspetto, il proprio stato e la propria parte logica;
- Permette una facile interoperabilità client-server;
- Può effettuare rendering sia sul client che sul server;
- Permette un aggiornamento efficiente della pagina quando cambia lo stato dei dati.

### 3.3.2 Svantaggi

- **Difficoltà di apprendimento:** imparare a sviluppare con React risulta essere difficoltoso nel periodo nel quale si ha il primo approccio con questa tecnologia;
- **Framework incompleto:** questo framework non comprende componenti fondamentali quali il router e altre librerie di gestione dei modelli.

### 3.4 LoopBack

*LoopBack*<sub>G</sub> è un framework di alto livello basato su Node.js che consente di creare API permettendo il collegamento con sorgenti di dati; esso è creato a partire da *Express*<sub>G</sub> e può, partendo da un modello di dati, generare facilmente delle *API*<sub>G</sub> REST completamente funzionanti che possono venire chiamate da un qualsiasi client.

#### 3.4.1 Vantaggi

- **Creazione di relazione tra modelli:** LoopBack consente di creare in maniera semplice delle relazioni tra modelli;
- **Supporto di più database:** LoopBack supporta molti database come ad esempio *MySQL*<sub>G</sub>, *Oracle*<sub>G</sub>, *SQL*<sub>G</sub> Server e MongoDB; inoltre è possibile collegare diversi modelli a diversi tipi di database facendoli poi interagire tra loro;
- **Semplice definizione dei ruoli:** LoopBack integra la possibilità di mettere in relazione modelli ed utenti per mezzo della definizione di *ACL*<sub>G</sub>(controlling data access);
- **Facilità di progettazione e implementazione:** essendo un'astrazione di Express, LoopBack consente di concentrarsi esclusivamente sulla logica che si vuole implementare evitando di focalizzare l'attenzione sulla progettazione di componenti quali, ad esempio, router.
- **Documentazione ufficiale:** chiara ed esaustiva.

#### 3.4.2 Svantaggi

- **Scarsa diffusione:** nonostante la buona documentazione ufficiale, non sono presenti molte guide ed esempi di applicazioni scritte con tale framework rendendo lo sviluppo più complicato;
- **Difficoltà di apprendimento:** a causa della presenza di molte componenti separate, la curva di apprendimento risulta essere piuttosto ripida.

### 3.5 HTML5

HTML5 è un linguaggio di markup per la strutturazione di pagine web, pubblicato dal *W3C*<sub>G</sub> nell'ottobre del 2014 con lo scopo di migliorare l'Html4, permettendo supporto a nuovi file multimediali, mantenendo la leggibilità da parte di uomini e computer.

#### 3.5.1 Vantaggi

- **Standard:** HTML5 è uno standard W3C.

### 3.5.2 Svantaggi

- **Scarso supporto ai vecchi browser:** HTML5 non è supportato da tutti i vecchi browser e sarà necessario utilizzare altre tecnologie di supporto per rimediare a tali mancanze.

## 3.6 CSS

$CSS_G$  (Cascading Style Sheets) è un linguaggio utile a descrivere la presentazione di un documento scritto in un linguaggio di markup; uno degli scopi principali che hanno spinto alla creazione di questo linguaggio è il bisogno di separare il contenuto del documento dalla sua presentazione rendendo così maggiormente accessibili le pagine dei siti web.

### 3.6.1 Vantaggi

- **Standard:** CSS è uno standard W3C;
- **Accessibilità:** la separazione tra contenuto e presentazione consente di avere pagine molto più accessibili rispetto ad avere tutto nello stesso file.

### 3.6.2 Svantaggi

- **Comportamento imprevedibile:** il comportamento delle regole CSS potrebbe variare a seconda del browser utilizzato.

## 3.7 Sweet.js

Sweet.js è un modulo che permette di definire delle macro; in particolare, una macro, è una porzione di codice DSL che una volta espansa, genera il codice javascript corrispondente ad essa. Con Sweet.js è possibile modellare un linguaggio ad hoc basato su JavaScript, collezionare le macro in un singolo modulo e condividerlo sul Node Packaged Modules; in MaaS esso viene utilizzato per implementare delle operazioni associate alle keyword di un linguaggio di dominio, il quale permetterà agli utenti di definire delle particolari operazioni sui dati contenuti nei propri database.

### 3.7.1 Vantaggi

- **Semplicità:** grazie a Sweet.js è possibile creare un linguaggio molto semplice.

### 3.7.2 Svantaggi

- **Documentazione:** non è presente molta documentazione e quella presente, è di scarsa qualità in quanto incompleta.

## 4 Descrizione architetturale

### 4.1 Metodo e formalismo di specifica

Le scelte progettuali, adottate nello sviluppo di MaaS, sono state profondamente influenzate dalle tecnologie utilizzate.

In primo luogo il progetto è basato su Node.js ed è scritto quindi in JavaScript: un linguaggio che è orientato agli oggetti ( $OOP_G$ ), ma che lascia grande libertà al programmatore nella scelta della tecnica da utilizzare per l'implementazione di pattern come l'*incapsulamento<sub>G</sub>* e l'*ereditarietà<sub>G</sub>*. Al contrario di altri linguaggi (C++, Java e derivati) non c'è un costrutto esplicito con il quale il programmatore può definire classi.

Progettare il sistema con un'architettura ad oggetti classica non permette di rappresentare in modo naturale la gestione dinamica dei tipi e le caratteristiche tipiche degli stili di programmazione funzionali.

Per rappresentare l'utilizzo delle funzioni come parametro tipico della programmazione funzionale è stato necessario valutare se rappresentarlo come classe o se utilizzare una notazione particolare, dato che i diagrammi delle classi UML si adattano poco all'utilizzo di codice proveniente dalla programmazione funzionale. La prima opzione avrebbe richiesto di raddoppiare quasi il numero di classi progettate, quindi con l'intenzione di mantenere la specifica tecnica chiara e maneggevole si è scelto di utilizzare una notazione ad hoc. Tale notazione è della forma “`function(<parametri>)`” e rappresenta il tipo di dato di una funzione che richiede i parametri “`<parametri>`”.

Il nostro approccio alla progettazione è stato contemporaneamente top-down e bottom-up. Da un lato siamo partiti suddividendo il sistema in front-end e back-end, definendo l'interfaccia di comunicazione, scegliendo di seguire in ciascuno l'organizzazione suggeritaci dai framework (LoopBack e React). Dall'altro lato siamo partiti dal basso, componendo e cercando di riutilizzare il più possibile le librerie già esistenti. Per far questo abbiamo prima cercato e confrontato con attenzione la struttura e le scelte sia di progetti open source che di progetti proposti come best practice.

L'approccio top-down è stato schematizzato nei diagrammi di deployment e dei package.

Per descrivere il sistema sono stati utilizzati i diagrammi di sequenza e di attività, descrivendo l'interazione tra i singoli oggetti. In questo modo siamo riusciti a descrivere alcuni dei meccanismi tipici dell'applicazione, in particolar modo l'ordine in cui agiscono i *moduli<sub>G</sub>* di LoopBack. Essi saranno molto utili per la progettazione di dettaglio e per la codifica.

I diagrammi di deployment, dei package, delle classi, di sequenza e di attività presentati di seguito utilizzano la specifica  $UML_G$  2.0, incrementata con la convenzione per la rappresentazione delle funzioni. Nei diagrammi dei package in particolare, quelli colorati in verde rappresentano librerie esterne dei framework.

Nella descrizione degli attributi, nel momento in cui viene usato il tipo di ritorno `JSFile`, si intende l'inclusione mediante comando `require(Module)`, di un file Javascript, con lo scopo di creare un riferimento ad esso, in modo da poter accedere ai suoi metodi.

Nel caso in cui sia presente un JSON nella segnatura di un metodo, esso andrà specificato seguendo la sintassi `JSON=Campo1, Campo2`.

## 4.2 Architettura generale

L'architettura di MaaS si suddivide in due componenti interne principali e una esterna. La prima componente interna, Client, rappresenta il browser degli utenti che usufruiscono del servizio MaaS. Essi interagiranno con il front-end dell'applicazione. La seconda componente, WebServer, rappresenta il server su cui viene implementato il back-end. Entrambe le componenti interagiranno tra loro, utilizzando delle API messe a disposizione dal back-end, che andranno a rispettare i principi REST. La componente esterna, External Database Server, rappresenta i server su cui vengono ospitati i database esterni MongoDB. La configurazione di tale componente non è compito del progetto, in quanto all'applicativo MaaS interessa solo conoscere i dati di connessione dei database esterni, in modo da interfacciarsi con essi in modalità sola lettura.

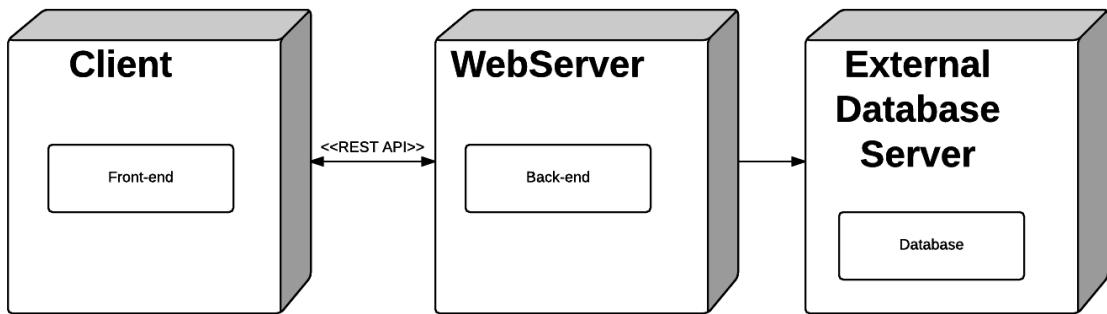


Figura 1: Diagramma del Deployment

#### 4.2.1 Back-end

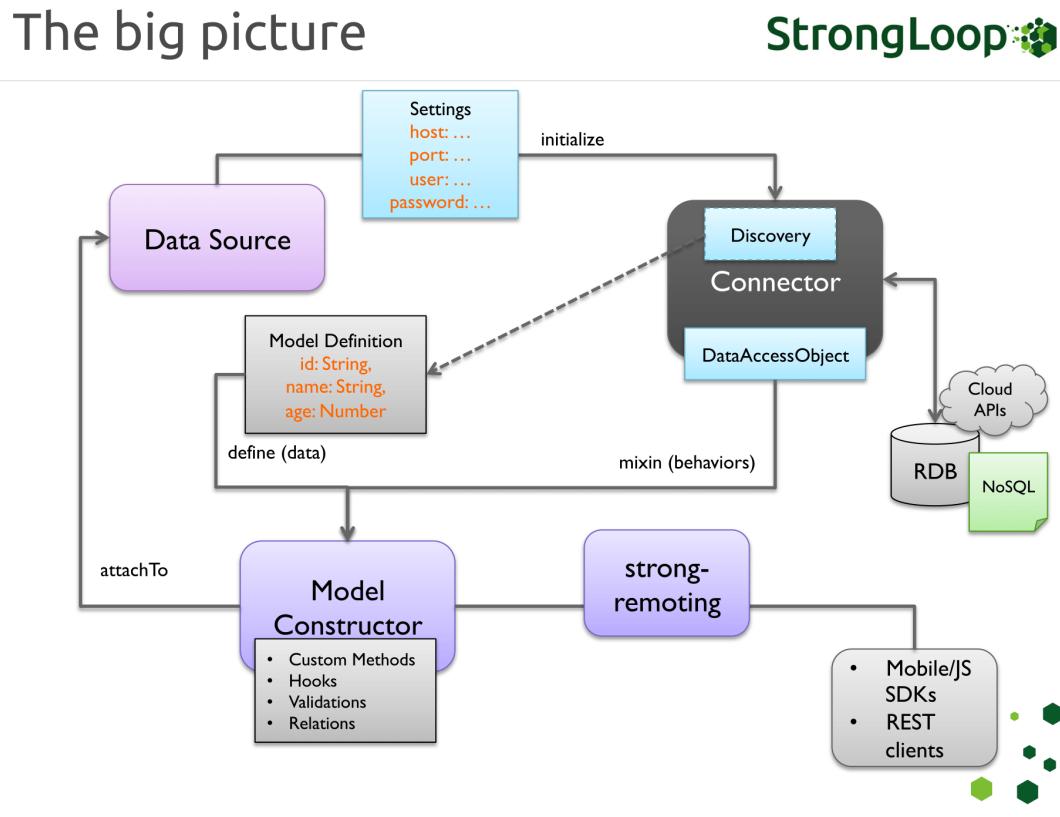


Figura 2: Architettura di un'applicazione LoopBack

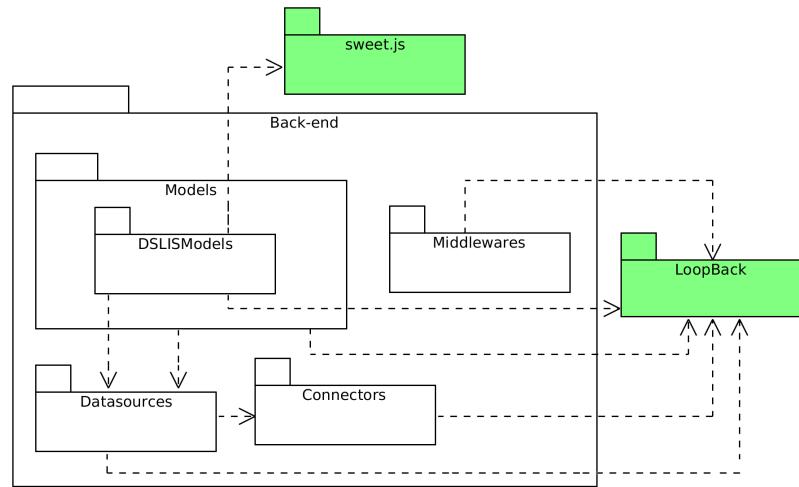


Figura 3: Diagramma dei package del back-end

Le comunicazioni tra il back-end e il front-end avvengono tramite scambio di messaggi HTTP e file in formato JSON.

#### 4.2.2 Front-end

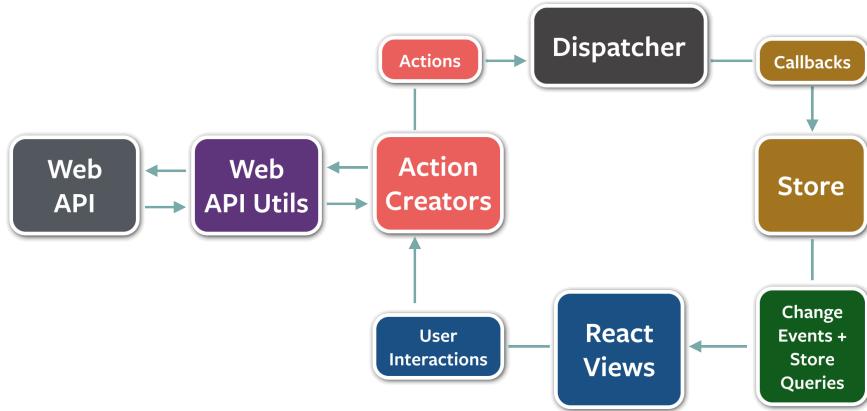


Figura 4: Architettura Flux

Il front-end è gestito dal framework React, la cui architettura è Flux<sup>1</sup>, che si pone come alternativa al classico MVC.

<sup>1</sup>Si veda appendice A.1.1 per approfondimenti

### 4.3 Interfaccia REST

Per la realizzazione del back-end si è adottata una interfaccia che cerca di aderire il più possibile ai principi REST. I vantaggi che ci hanno portato a questa scelta sono:

- indipendenza dai linguaggi di programmazione;
- indipendenza dalle piattaforme software;
- utilizzo semplice in presenza di firewall;
- semplicità di utilizzo.

I componenti chiave di un'architettura REST sono:

- **Risorse:** sono identificate da URL logici. Lo stato dell'applicazione e le funzionalità sono divisi in risorse web;
- **Rete di risorse:** una singola risorsa non dovrebbe essere sovradimensionata e contenere dettagli troppo a grana fine;
- **Client-server:** un insieme di interfacce uniforme separa i client dai server. Server e client possono essere sostituiti e sviluppati indipendentemente fintanto che l'interfaccia non viene modificata;
- **Stateless:** la comunicazione client-server è ulteriormente vincolata in modo che nessun contesto client venga memorizzato sul server tra le richieste. Ogni client contiene tutte le informazioni necessarie per richiedere il servizio, e lo stato della sessione è contenuto sul client;
- **Cacheable:** come nel World Wide Web, i client possono fare caching delle risposte. Le risposte devono in ogni modo definirsi implicitamente o esplicitamente cacheable o no, in modo da prevenire che i client possano riusare stati vecchi o dati errati. Una gestione ben fatta della cache può ridurre o parzialmente eliminare le comunicazioni client server, migliorando scalabilità e performance;
- **Layered system:** un client non può dire se è connesso direttamente ad un server di livello più basso od intermedio, i server intermedi possono migliorare la scalabilità del sistema con load-balancing o con cache distribuite. Layer intermedi possono offrire inoltre politiche di sicurezza.

La seguente tabella mostra le relazioni tra gli URI ed i metodi HTTP:

Risorsa	URI della collection es. <a href="http://example.com/resources/">http://example.com/resources/</a>	URI di un elemento es. <a href="http://example.com/resources/item17">http://example.com/resources/item17</a>
<b>GET</b>	<b>Fornisce</b> informazioni sui membri della collection.	<b>Fornisce</b> una rappresentazione dell'elemento della collection indicato, espresso in un appropriato formato.
<b>PUT</b>	Non usata.	<b>Sostituisce</b> l'elemento della collection indicato, o se non esiste, lo <b>crea</b> .
<b>POST</b>	<b>Crea</b> un nuovo elemento nella collection. La URI del nuovo elemento è generata automaticamente ed è di solito restituita dall'operazione.	Non usato.
<b>DELETE</b>	Non usata.	<b>Cancella</b> l'elemento della collection indicato.

I dati vengono rappresentati adottando il formato JSON, in quanto il suo utilizzo tramite JavaScript è particolarmente semplice.

## 5 Back-end

### 5.1 Descrizione packages e classi

#### 5.1.1 Back-end

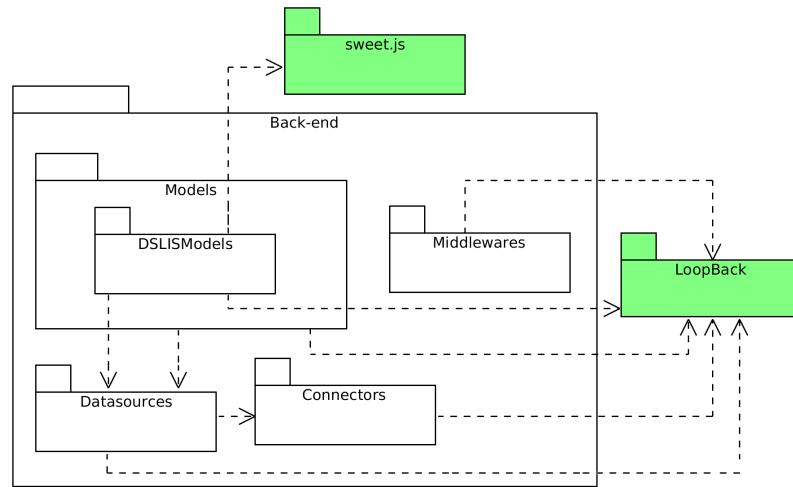


Figura 5: Diagramma dei package del back-end

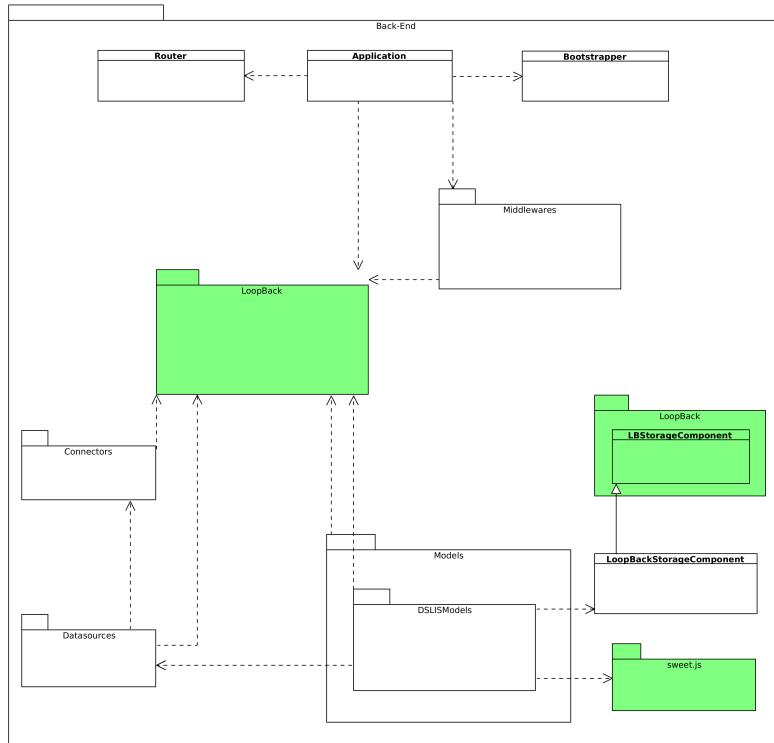


Figura 6: Diagramma delle classi del back-end

#### 5.1.1.1 Informazioni sul package

**Descrizione** Package che racchiude tutte le componenti del Back-end. Comprende tutte le classi che rappresentano la parte server dell'applicativo.

Le componenti sono organizzate secondo il design pattern previsto per il framework node.js LoopBack.

#### Package contenuti

- Models
- Connectors
- Datasources
- Middlewares

#### Framework esterni

- LoopBack
- sweet.js

### 5.1.1.2 Classi

#### Application

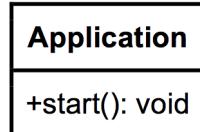


Figura 7: Diagramma della classe Application

#### Descrizione

Classe che rappresenta la parte back-end dell'applicazione. Viene derivata dal framework LoopBack. Inoltre, rappresenta l'elemento client del design pattern Chain of Responsibility<sup>2</sup>.

#### Utilizzo

Viene utilizzata come elemento "main", per inizializzare le componenti del back-end. In particolare, si occupa di iniziare una richiesta ad un oggetto concreto della classe Back-end::Middlewares::MiddlewareHandler.

#### Relazioni con altre classi

- Back-end::Middlewares::MiddlewareHandler
- Back-end::Bootstrapper

#### Attributi

Assenti.

#### Metodi

`+start(): void`

Questo metodo carica l'oggetto di configurazione e fa partire il server dell'applicazione.

#### Bootstrapper

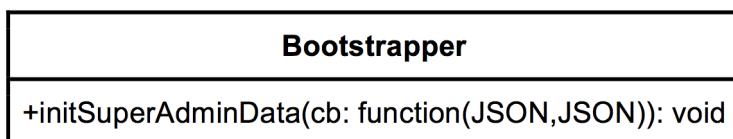


Figura 8: Diagramma della classe Bootstrapper

<sup>2</sup>Si veda appendice A.3.3 per approfondimenti

## Descrizione

Classe che rappresenta gli script di boot eseguiti all'avvio dell'applicazione. Viene utilizzato il pattern Module<sup>3</sup>, per dare visibilità o meno ai metodi.

## Utilizzo

Viene utilizzata per fornire i metodi utili ad inizializzare l'applicazione, come le funzioni per configurare le impostazioni dell'applicazione ed aggiungere i dati che consentono l'avvio dell'applicazione. In particolare, inserisce i dati relativi al Super Amministratore nel database del sistema MaaS.

## Attributi

Assenti.

## Metodi

`+initSuperAdminData(cb: function(JSON,JSON)): void`

Questo metodo si occupa di inizializzare i dati e le configurazioni dei Super Amministratori della piattaforma MaaS;

`- cb: function(JSON, JSON)`

Rappresenta la callback che il metodo deve chiamare al termine della configurazione. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

## LoopBackStorageComponent

## Descrizione

Classe che fornisce i metodi utili ad effettuare l'upload e il download dei file. Viene ereditata dal framework LoopBack.

## Utilizzo

Viene utilizzato per eseguire l'upload e il download dei file.

## Classi ereditate

- LoopBack::LBStorageComponent

## Attributi

Assenti.

---

<sup>3</sup>Si veda appendice A.2.1 per approfondimenti

## Metodi

Assenti, in quanto utilizza i metodi forniti dal framework LoopBack.

### 5.1.2 Back-end::Models

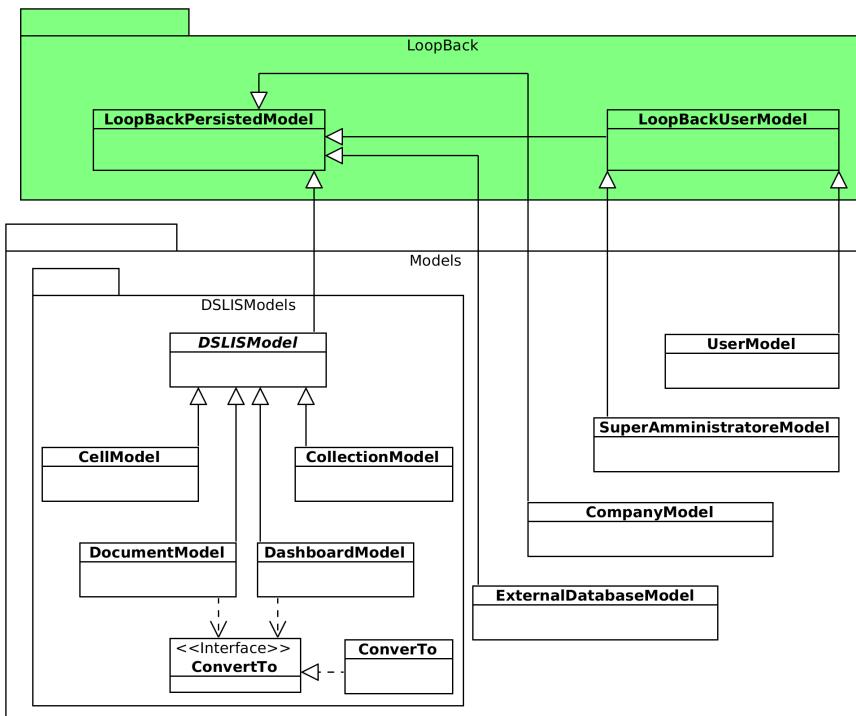


Figura 9: Diagramma delle classi del package Models

#### 5.1.2.1 Informazioni sul package

##### Descrizione

Package contenente le componenti che rappresentano e gestiscono le risorse dei dati di back-end, come database, REST API e altri servizi di back-end. Ogni Model include le REST API utili a ricevere le richieste del client ed a mapparle con le relative funzioni. Viene utilizzato il design pattern Module per dare visibilità o meno ai metodi.

##### Package contenuti

- DSLISModels

##### Framework esterni

- LoopBack

- sweet.js

### 5.1.2.2 Classi

#### UserModel

UserModel
-userData : JSON +createUser(newUser : JSON, cb : function(JSON, JSON)) : void +loginUser(user : JSON, cb : function(JSON, JSON)) : void +editUser(user : JSON, cb : function(JSON, JSON)) : void +logoutUser(user : JSON, cb : function(JSON, JSON)) : void +getUsers(cb : function(JSON, JSON)) : void +getUser(email : string, cb : function(JSON, JSON)) : void +deleteUser(email : string, cb : function(JSON, JSON)) : void +searchUser(value : JSON, cb : function(JSON, JSON)) : void +forgotPassword(email : string, cb : function(JSON, JSON)) : void +changeRole(user : JSON, role : JSON, cb : function(JSON, JSON)) : void +sendInvite(email : string, cb : function(JSON, JSON)) : void

Figura 10: Diagramma della classe UserModel

#### Descrizione

Classe che rappresenta la logica dell'applicazione nell'ambito degli utenti. Utilizza il design pattern Module per dare visibilità o meno ai metodi.

#### Utilizzo

Viene utilizzata per fornire le proprietà, gli ACL (Access Control Lists), le relazioni, i ruoli ed i metodi riguardanti l'ambito degli utenti. La classe sarà relazionata con il database MongoDB dell'applicazione, rappresentato dalla classe Back-end::Datasources::DatabaseDatasource.

#### Classi ereditate

- LoopBack::LoopBackUserModel

#### Relazioni con altre classi

- Back-end::Datasources::EmailDatasource
- Back-end::Datasources::DatabaseDatasource
- Back-end::LoopBackStorageComponent

## Attributi

### **-userData: JSON**

Rappresenta i dati di un Utente in formato JSON.

- email di tipo *string*
- password di tipo *string*
- name di tipo *string*
- surname di tipo *string*
- birthDate di tipo *Date*
- gender di tipo *enum*

## Metodi

### **+createUser(newUser: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di inserire un utente nel database interno di MaaS.

#### **– newUser: JSON**

Rappresenta i dati utente da utilizzare nella creazione di un nuovo utente;

#### **– cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della registrazione di un utente su MaaS. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON. In caso di esito negativo il JSON mostra una stringa rappresentante l'insuccesso dell'operazione, mentre per l'esito positivo sono mostrati l'id access token, il time to live, la data di creazione e l'id dell'utente.

### **+loginUser(user: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di gestire l'autenticazione di un utente su MaaS.

#### **– user: JSON**

Rappresenta i dati utente da utilizzare per effettuare l'autenticazione di un utente;

#### **– cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'autenticazione di un utente su MaaS. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente l'id e l'email dell'utente in caso di successo, mentre per l'esito negativo contiene una stringa rappresentante l'errore.

### **+editUser(user: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di effettuare la modifica di un utente su MaaS.

#### **– user: JSON**

Rappresenta i nuovi dati di un utente che andranno a rimpiazzare quelli già esistenti;

#### **– cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della modifica dei dati di un utente su MaaS. Nel caso di errori server verrà ritornato un

JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+logoutUser(user: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di effettuare disconnessione di un utente da MaaS.

**- user: JSON**

Rappresenta i nuovi dati dell'utente che effettuerà la disconnessione dal servizio;

**- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine del logout di un utente da MaaS. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+getUsers(company: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si ritornare i dati relativi agli utenti di una determinata azienda.

**- company: JSON**

Rappresenta i dati dell'azienda di cui si vuole ottenere la lista degli utenti registrati;

**- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della richiesta dati degli utenti iscritti su una azienda. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente i dati di tutti gli utenti in caso di successo, mentre per l'esito negativo una stringa rappresentante l'errore.

**+getUser(email: string, cb: function(JSON, JSON)) : void**

Questo metodo si ritornare i dati relativi ad un utente registrato su MaaS.

**- email: string**

Rappresenta la email dell'utente di cui si vogliono ottenere i dati;

**- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della richiesta dati di un utente MaaS. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente i dati dell'utente in caso di successo, mentre per l'esito negativo una stringa rappresentante l'errore.

**+deleteUser(email: string, cb: function(JSON, JSON)) : void**

Questo metodo effettua l'eliminazione di un utente registrato su MaaS.

**- email: string**

Rappresenta la email dell'utente che si vuole eliminare;

**- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'eliminazione di un utente MaaS. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+searchUser(value: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di effettuare una ricerca. Sarà restituita una lista di tutti gli utenti che coincidono con i valori richiesti.

**- value: JSON**

Rappresenta l'insieme delle caratteristiche che devono rispettare gli utenti nella lista ritornata;

**- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della ricerca filtrata degli utenti su MaaS. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente i dati degli utenti che matchano con la ricerca in caso di successo, mentre per l'esito negativo una stringa rappresentante l'errore.

**+forgotPassword(email: string, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di effettuare la procedura di ripristino password per un utente MaaS.

**- email: string**

Rappresenta l'email dell'utente che vuole ripristinare la propria password;

**- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della procedura ripristino password per un utente MaaS. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+changeRole(user: JSON, role: JSON, cb: function(JSON, JSON)) : void**

Questo metodo permette il cambio ruolo un utente MaaS.

**- user: JSON**

Rappresenta i dati dell'utente a cui si vuole cambiare il ruolo;

**- role: JSON**

Rappresenta i dati relativi al ruolo che viene assegnato all'utente user.

**- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della procedura di modifica ruolo per un utente MaaS. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+sendInvite(email: string, role: JSON, cb: function(JSON, JSON)) : void**

Questo metodo permette l'invito di un utente su MaaS trami invio email.

**- email: string**

Rappresenta l'email dell'utente invitato su MaaS.

**- role: JSON**

Rappresenta i dati relativi al ruolo che viene assegnato all'utente invitato.

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della procedura di invito tramite mail su MaaS. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

### SuperAmministratoreModel

SuperAmministratoreModel	
-superAdminData : JSON	
+loginSuperAdmin(superAdmin : JSON, cb : function(JSON, JSON)) : void	
+logoutSuperAdmin(superAdmin : JSON, cb : function(JSON, JSON)) : void	
+impersonateUser(user : JSON, superAdmin : JSON, cb : function(JSON, JSON)) : void	

Figura 11: Diagramma della classe SuperAmministratoreModel

### Descrizione

Classe che rappresenta la logica dell'applicazione nell'ambito del Super Amministratore. Utilizza il design pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per fornire le proprietà, gli ACL (Access Control Lists), le relazioni, i ruoli ed i metodi riguardanti l'ambito del Super Amministratore. La classe sarà relazionata con il database MongoDB dell'applicazione, rappresentato dalla classe Back-end::Datasources::DatabaseDatasource.

### Classi ereditate

- LoopBack::LoopBackUserModel

### Relazioni con altre classi

- Back-end::Datasources::DatabaseDatasource

### Attributi

**-superAdminData: JSON**

Rappresenta i dati di un Super Amministratore in formato JSON.

- email di tipo *string*
- password di tipo *string*

## Metodi

**+loginSuperAdmin(superAdmin: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di autenticare un Super Amministratore alla piattaforma, ricevendo le sue credenziali e controllando che siano corrette.

– **superAdmin: JSON**

Rappresenta i dati del Super Amministratore da utilizzare nella sua autenticazione;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'elaborazione, Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente l'id e l'email del super amministratore in caso di successo, mentre per l'esito negativo contiene una stringa rappresentante l'errore.

**+logoutSuperAdmin(superAdmin: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di disconnettere un Super Amministratore dalla piattaforma, ricevendo il suo identificativo.

– **superAdmin: JSON**

Rappresenta i dati del Super Amministratore da utilizzare nella sua disconnessione per identificarlo;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'elaborazione, Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+impersonateUser(user: JSON, superAdmin: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di impersonificare un Super Amministratore in un utente iscritto a MaaS per intervenire in caso di necessità offrendo supporto attraverso il profilo interessato.

– **user: JSON**

Rappresenta i dati dell'utente da impersonificare, utili per identificarlo;

– **superAdmin: JSON**

Rappresenta i dati del Super Amministratore da utilizzare nell'impersonificazione per identificarlo;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'elaborazione, Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

## CompanyModel

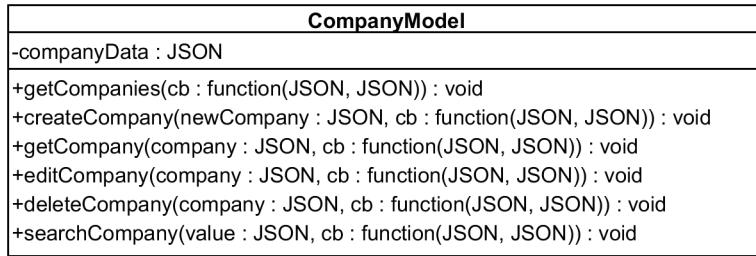


Figura 12: Diagramma della classe CompanyModel

### Descrizione

Classe che rappresenta la logica dell'applicazione nell'ambito delle aziende.  
Utilizza il design pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per fornire le proprietà, gli ACL (Access Control Lists), le relazioni, i ruoli ed i metodi riguardanti l'ambito delle aziende. La classe sarà relazionata con il database MongoDB dell'applicazione, rappresentato dalla classe Back-end::Datasources::DatabaseDatasource.

### Classi ereditate

- LoopBack::LoopBackPersistedModel

### Relazioni con altre classi

- Back-end::Datasources::DatabaseDatasource

### Attributi

**-companyData: JSON**

Rappresenta i dati di un'azienda iscritta alla piattaforma in formato JSON.

- name di tipo *string*

### Metodi

**+getCompanies(cb: function(JSON, JSON)) : void**

Questo metodo si occupa di ritornare la lista delle aziende ad un Super Amministratore;

**– cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'elaborazione della lista delle aziende. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente i dati di tutte le aziende in caso di successo, mentre per l'esito negativo una stringa rappresentante l'errore.

**+createCompany(newCompany: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di creare un'azienda e di registrarla nel database di MaaS;

- **newCompany: JSON**

Rappresenta i dati dell'azienda da utilizzare nella sua creazione;

- **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della creazione di un'azienda. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+getCompany(company: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di ritornare un'azienda ad un Super Amministratore;

- **company: JSON**

Rappresenta i dati dell'azienda da ritornare, utili per identificarla;

- **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine del ritorno di un'azienda. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente i dati dell'azienda in caso di successo, mentre per l'esito negativo una stringa rappresentante l'errore.

**+editCompany(company: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di modificare i dati di un'azienda da parte di un Super Amministratore;

- **company: JSON**

Rappresenta i dati dell'azienda da modificare, utili per identificarla;

- **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della modifica dei dati di un'azienda. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+deleteCompany(company: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di eliminare un'azienda da parte di un Super Amministratore;

- **company: JSON**

Rappresenta i dati dell'azienda da eliminare, utili per identificarla;

- **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'eliminazione un'azienda. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+searchCompany(value: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di cercare un'azienda da parte di un Super Amministratore;

- **value: JSON**

Rappresenta i dati utili per identificare l'azienda da cercare nel database;

**- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della ricerca di un'azienda. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente i dati di tutte le aziende che matchano con la ricerca in caso di successo, mentre per l'esito negativo una stringa rappresentante l'errore.

### ExternalDatabaseModel

ExternalDatabaseModel	
-	externalDatabaseData : JSON
+	addSource(database : JSON, cb : function(JSON, JSON)) : void
+	deleteSource(database : JSON, cb : function(JSON, JSON)) : void
+	getData(query : JSON, cb : function(JSON, JSON)) : void
+	searchDB(value : JSON, company : JSON, cb : function(JSON, JSON)) : void
+	allowAccess(user : JSON, database : JSON, cb : function(JSON, JSON)) : void
+	denyAccess(user : JSON, database : JSON, cb : function(JSON, JSON)) : void

Figura 13: Diagramma della classe ExternalDatabaseModel

### Descrizione

Classe che rappresenta la logica dell'applicazione nell'ambito dei database MongoDB esterni. Utilizza il design pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per fornire le proprietà, gli ACL (Access Control Lists), le relazioni, i ruoli ed i metodi riguardanti l'ambito dei database MongoDB esterni. La classe sarà relazionata con il database MongoDB dell'applicazione, rappresentato dalla classe Back-end::Datasources::DatabaseDatasource.

### Classi ereditate

- LoopBack::LoopBackPersistedModel

### Relazioni con altre classi

- Back-end::Datasources::DatabaseDatasource

### Attributi

**-externalDatabaseData: JSON**

Ovvero i dati di una risorsa che rappresenta un database collegato esternamente da parte dell'azienda.

- company di tipo *string* che rappresenta l'azienda al quale appartiene il database esterno;
- connect di tipo *string* che rappresenta la stringa di connessione al database esterno.

## Metodi

**+addSource(database: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di creare una risorsa rappresentante un database esterno collegato al sistema MaaS da parte degli amministratori di un'azienda;

- **database: JSON**

Rappresenta il database esterno da collegare;

- **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della creazione del nuovo database esterno. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+deleteSource(database: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di eliminare una risorsa rappresentante un database esterno collegato al sistema MaaS da parte degli amministratori di un'azienda;

- **database: JSON**

Rappresenta il database esterno da scollegare;

- **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della rimozione del database esterno. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+getData(query: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di mostrare determinati dati di un database esterno collegato al sistema MaaS da parte degli amministratori di un'azienda;

- **query: JSON**

Rappresenta la query da eseguire sul database, ovvero la restrizione o filtro che andrà a selezionare determinati valori, compresa l'azienda che possiede il database;

- **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine del ritorno dei dati. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente i dati richiesti in caso di successo, mentre per l'esito negativo una stringa rappresentante l'errore.

**+searchDB(value: JSON, company: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di cercare un database esterno collegato al sistema MaaS da parte degli amministratori di un'azienda;

- **value: JSON**

Rappresenta i valori dei criteri di ricerca;

**- company: JSON**

Rappresenta l'azienda che possiede il database esterno;

**- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della ricerca del database. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente i dati dei database che matchano con la ricerca in caso di successo, mentre per l'esito negativo una stringa rappresentante l'errore.

**+allowAccess(user: JSON, database: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di concedere i permessi di accesso ad un database esterno collegato al sistema MaaS ad un utente dell'azienda;

**- user: JSON**

Rappresenta i dati dell'utente al quale concedere i permessi di accesso;

**- database: JSON**

Rappresenta i dati di un database esterno di cui concedere i permessi di accesso;

**- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della modifica dei permessi al database. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+denyAccess(user: JSON, database: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di negare i permessi di accesso ad un database esterno collegato al sistema MaaS ad un utente dell'azienda.

**- user: JSON**

Rappresenta i dati dell'utente al quale negare i permessi di accesso;

**- database: JSON**

Rappresenta i dati di un database esterno di cui negare i permessi di accesso;

**- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della modifica dei permessi al database. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

### 5.1.3 Back-end::Models::DSLISModels

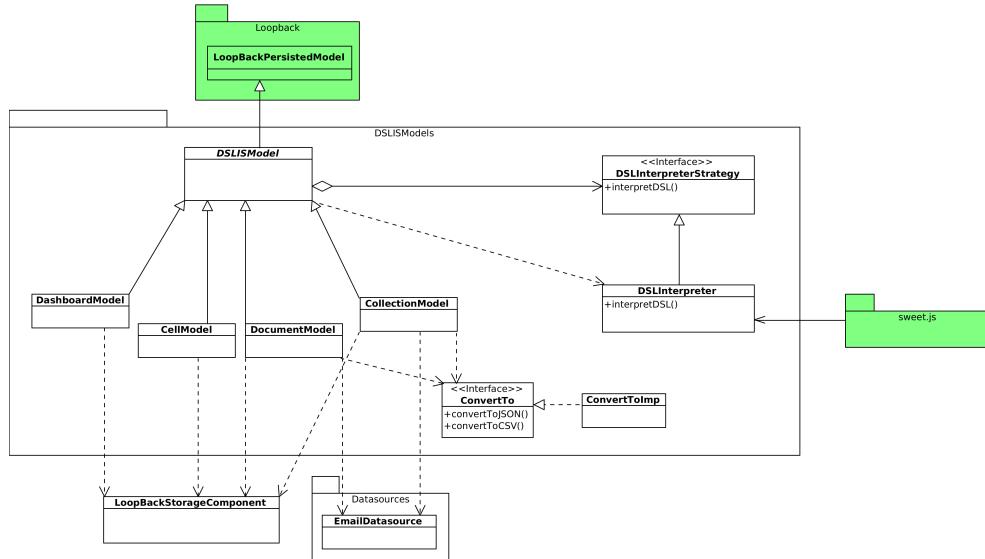


Figura 14: Diagramma delle figure del package: Back-end::Models::DSLISModels

#### 5.1.3.1 Informazioni sul package

##### Descrizione

Package contenente le classi che rappresentano e gestiscono le risorse dei dati di back-end nell'ambito dei DSLIS. Ogni Model include le REST API utili a ricevere le richieste del client ed a mapparle con le relative funzioni.

Viene utilizzato il design pattern Module per dare visibilità o meno ai metodi.

##### Framework esterni

- LoopBack
- sweet.js

#### 5.1.3.2 Classi

##### DSLISModel

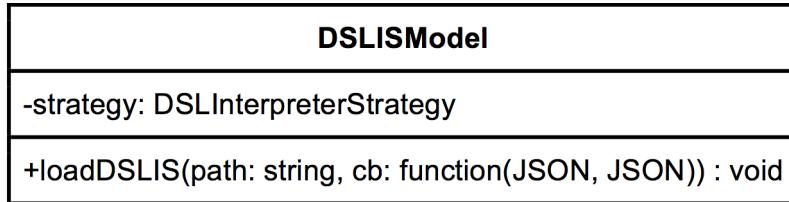


Figura 15: Diagramma della classe DSLISModel

### Descrizione

Classe base astratta per i Model riguardanti i DSLIS. Viene derivata dalla classe LoopBackPersistedModel fornita dal framework LoopBack.

### Utilizzo

Viene utilizzata come classe Context per utilizzare il metodo fornito dalla classe DSLInterpreterStrategy, appartenente al design pattern Strategy<sup>4</sup>, per interpretare il DSLIS con un algoritmo, nel caso di necessità intercambiabile.

### Estensioni

- Back-end::Models::DSLISModels::CollectionModel
- Back-end::Models::DSLISModels::CellModel
- Back-end::Models::DSLISModels::DocumentModel
- Back-end::Models::DSLISModels::DashboardModel

### Classi ereditate

- LoopBack::LoopBackPersistedModel

### Relazioni con altre classi

- Back-end::Models::DSLISModels::DSLInterpreter

### Attributi

**-strategy: DSLInterpreterStrategy**

Ovvero il riferimento alla classe strategy del design pattern Strategy.

---

<sup>4</sup>Si veda appendice A.3.2 per approfondimenti

## Metodi

```
+loadDSLIS(path: string, cb: function(JSON, JSON)) : void
```

Questo metodo si occupa di richiamare l'interprete per il caricamento del codice DSL;

- **path: string**

Rappresenta la locazione del codice DSL da caricare;

- **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine del caricamento.

Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

## ConvertTo

### **ConvertTo**

```
+convertToJson(DSLIS: JS Object, cb: function(JSON, JSON)) : void
+convertToCSV(DSLIS: JS Object, cb: function(JSON, CSV)) : void
```

Figura 16: Diagramma della classe ConvertTo

## Descrizione

Interfaccia che fornisce i metodi utili ad effettuare la conversione di un oggetto Javascript in un JSON oppure in un CSV, seguendo il design pattern Constructor Injection<sup>5</sup> per diminuire le dipendenze.

## Utilizzo

Viene chiamata da Back-end::Models::DSLISModels::DocumentModel e Back-end::Models::DSLISModels::CollectionModel per convertire la struttura dei dati rappresentati da queste entità in dei formati predefiniti.

## Estensioni

- Back-end::Models::DSLISModels::ConvertToImp

## Attributi

Assenti.

---

<sup>5</sup>Si veda appendice A.1.2 per approfondimenti

## Metodi

**+convertToJson(DSLIS: JS Object, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di convertire in formato JSON un JS Object contenente il codice del DSLIS.

– **DSLIS: JS Object**

Rappresenta il codice del DSLIS;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della conversione.

Se non ci sono errori verrà ritornato un file JSON contenente il codice del DSLIS.

Nel caso di errori server verrà ritornato un JSON.

**+convertToCSV(DSLIS: JS Object, cb: function(JSON, CSV))**

Questo metodo si occupa di convertire in formato CSV un JS Object contenente il codice del DSLIS.

– **DSLIS: JS Object**

Rappresenta il codice del DSLIS;

– **cb: function(JSON, CSV)**

Rappresenta la callback che il metodo deve chiamare al termine della conversione.

Se non ci sono errori verrà ritornato un file CSV contenente il codice del DSLIS.

Nel caso di errori server verrà ritornato un JSON.

## ConverToImp

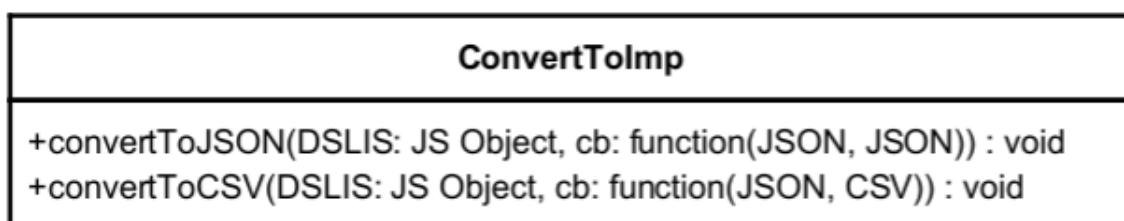


Figura 17: Diagramma della classe ConvertToImp

## Descrizione

Classe che implementa ConvertTo per fornire i metodi utili ad effettuare la conversione di un oggetto Javascript in un JSON oppure in un CSV.

## Utilizzo

Viene utilizzata da Back-end::Models::DSLISModels::DocumentModel e Back-end::Models::DSLISModels::Collection per convertire la struttura dei dati rappresentati da queste entità in dei formati predefiniti.

## Relazioni con altre classi

- Back-end::Models::DSLISModels::ConvertTo

### Classi ereditate

- Back-end::Models::DSLISModels::ConvertTo

### Attributi

Assenti.

### Metodi

```
+convertToJson(DSLIS: JS Object, cb: function(JSON, JSON)) : void
```

Questo metodo si occupa di implementare i metodi corrispondenti della classe ConvertTo;

```
+convertToCSV(DSLIS: JS Object, cb: function(JSON, CSV))
```

Questo metodo si occupa di implementare i metodi corrispondenti della classe ConvertTo.

### CollectionModel

<b>CollectionModel</b>
-collectionData : JSON -conv : ConvertTo  +extract(id : string, cb : function(JSON, JSON)) : void +CollectionModel(conv : ConvertTo, cb : function(JSON, JSON)) : CollectionModel +getCollections(cb : function(JSON, JSON)) : void +createCollection(newCollection : JSON, cb : function(JSON, JSON)) : void +editCollection(id : string, collection : JSON, cb : function(JSON, JSON)) : void +deleteCollection(id : string, cb : function(JSON, JSON)) : void +retrieveCollection(id : string, cb : function(JSON, JSON)) : void +searchCollection(value : JSON, cb : function(JSON, JSON)) : void +executeCollection(id : string, cb : function(JSON, JSON)) : void +sendEmail(id : string, format : string, cb : function(JSON, JSON)) : void +export(id : string, format : string, cb : function(JSON, JSON)) : void +exportDSLIS(id : string, cb : function(JSON, JSON)) : void +importDSLIS(collectionDSLIS : JSON, cb : function(JSON, JSON)) : void

Figura 18: Diagramma della classe CollectionModel

### Descrizione

Classe che rappresenta la logica dell'applicazione nell'ambito delle Collection, create mediante DSLIS.

Utilizza il design pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per fornire le proprietà, gli ACL (Access Control Lists), le relazioni, i ruoli ed i metodi riguardanti l'ambito delle Collection. La classe sarà relazionata con il database MongoDB dell'applicazione, rappresentato dalla classe Back-end::Datasources::DatabaseDatasource.

## Classi ereditate

- Back-end::Models::DSLISModels::DSLISModel

## Relazioni con altre classi

- Back-end::Models::DSLISModels::ConvertTo
- Back-end::Datasources::DatabaseDatasource
- Back-end::Datasources::EmailDatasource
- Back-end::LoopBackStorageComponent

## Attributi

### **-collectionData: JSON**

Rappresenta i dati di una Collection dell'azienda.

- id di tipo *string* che rappresenta l'identificativo della Collection;
- DLSIS tipo *string* che rappresenta il codice DSL della Collection.

### **conv: ConvertTo**

Rappresenta il riferimento alla classe ConvertTo per la conversione di una Collection in un determinato formato.

## Metodi

### **+CollectionModel(conv: ConvertTo, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di ritornare un JS Object estraendo i dati da una Collection.

#### **- conv: ConvertTo**

Rappresenta il riferimento alla classe ConvertTo per la conversione di una Collection in un determinato formato;

#### **- cb: function(JSON, JS Object)**

Rappresenta la callback che il metodo deve chiamare al termine della creazione. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

### **+getCollections(cb: function(JSON, JSON)) : void**

Questo metodo si occupa di ritornare la lista delle Collection a cui un utente ha accesso.

#### **- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della richiesta. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente i dati di tutte le Collections in caso di successo, mentre per l'esito negativo una stringa rappresentante l'errore.

### **+createCollection(newCollection: JSON, cb: function(JSON, JSON)) : void**

Questo metodo permette ad un utente MaaS di creare un una nuova Collection;

- **newCollection: JSON**  
Rappresenta i dati delle nuova Collection;
- **cb: function(JSON, JSON)**  
Rappresenta la callback che il metodo deve chiamare al termine della creazione della Collection. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

```
+editCollection(id: string ,collection: JSON, cb: function(JSON, JSON)) : void
```

Questo metodo si occupa di modificare i dati relativi ad una Collection presente in MaaS;

- **id: string**  
Rappresenta l'id della Collection da modificare;
- **collection: JSON**  
Rappresenta i nuovi dati della Collection che andaranno a rimpiazzare quelli già esistenti. della;
- **cb: function(JSON, JSON)**  
Rappresenta la callback che il metodo deve chiamare al termine della modifica dei dati della Collection. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

```
+deleteCollection(id: string, cb: function(JSON, JSON)) : void
```

Questo metodo si occupa di eliminare una Collection presente su MaaS;

- **id: string**  
Rappresenta l'identificativo della Collection;
- **cb: function(JSON, JSON)**  
Rappresenta la callback che il metodo deve chiamare al termine dell'eliminazione della Collection. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

```
+retrieveCollection(id: string, cb: function(JSON, JSON)) : void
```

Questo metodo si occupa ritornare il codice di una Collection;

- **id: string**  
Rappresenta l'identificativo della Collection da modificare;
- **cb: function(JSON, JSON)**  
Rappresenta la callback che il metodo deve chiamare al termine della richiesta. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

```
+searchCollection(value: JSON, email: string, cb: function(JSON, JSON)) : void
```

Questo metodo si occupa richiedere una lista di Collection filtrando i risultati per valore;

**- `value: JSONs`**

Rappresenta i dati dell'utente al quale negare i permessi di accesso;

**- `cb: function(JSON, JSON)`**

Rappresenta la callback che il metodo deve chiamare al termine della ricerca filtrata per valori. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente i dati di tutte le Collection che matchano con la ricerca in caso di successo, mentre per l'esito negativo una stringa rappresentante l'errore.

**+`sendEmail(id: string, format: string, cb: function(JSON, JSON)) : void`**

Questo metodo si occupa inviare via mail i dati visualizzati in una Collection;

**- `id: string`**

Rappresenta l'identificativo della Collection da eseguire;

**- `format: string`**

Rappresenta il formato scelto per la Collection da inviare.

**- `cb: function(JSON, JSON)`**

Rappresenta la callback che il metodo deve chiamare al termine dell'invio mail. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+`export(id: string, format: string, cb: function(JSON, JSON)) : void`**

Questo metodo si occupa esportare i dati visualizzati in una Collection;

**- `id: string`**

Rappresenta l'identificativo della Collection da eseguire;

**- `format: string`**

Rappresenta il formato scelto per la Collection da inviare.

**- `cb: function(JSON, JSON)`**

Rappresenta la callback che il metodo deve chiamare al termine dell'esportazione della Collection. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+`exportDSLIS(id: string, cb: function(JSON, JSON)) : void`**

Questo metodo si occupa esportare il codice di una Collection;

**- `id: string`**

Rappresenta l'identificativo della Collection da esportare;

**- `cb: function(JSON, JSON)`**

Rappresenta la callback che il metodo deve chiamare al termine dell'esportazione del codice della Collection. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

```
+importDSLIS(collectionDSLIS: JSON, cb: function(JSON, JSON)) : void
```

Questo metodo si occupa importare il codice di una Collection;

- **collectionDSLIS: JSON**

Rappresenta il codice della Collection da importare;

- **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'importazione del codice della Collection. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

```
extract(id: string, cb: function(JSON, JS Object))
```

Questo metodo si occupa di ritornare un JS Object estraendo i dati da un Collection.

- **id: string**

Rappresenta l'identificativo della Collection;

- **cb: function(JSON, JS Object)**

Rappresenta la callback che il metodo deve chiamare al termine dell'estrazione. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

## DocumentModel

DocumentModel
-documentData : JSON
-conv : ConvertTo
+extract(id : string, cb : function(JSON, JSON)) : void
+DocumentModel(conv : ConvertTo, cb : function(JSON, JSON)) : DocumentModel
+getDocuments(cb : function(JSON, JSON)) : void
+createDocument(newDocument : JSON, cb : function(JSON, JSON)) : void
+editDocument(id : string, document : JSON, cb : function(JSON, JSON)) : void
+deleteDocument(id : string, cb : function(JSON, JSON)) : void
+searchDocument(value : JSON, email : string, cb : function(JSON, JSON)) : void
+executeDocument(id : string, cb : function(JSON, JSON)) : void
+executeDocument(collection : JSON, id : string, cb : function(JSON, JSON)) : void
+sendEmail(id : string, format : string, cb : function(JSON, JSON)) : void
+export(id : string, format : string, cb : function(JSON, JSON)) : void
+retrieveDocument(id : string, cb : function(JSON, JSON)) : void
+exportDSLIS(id : string, cb : function(JSON, JSON)) : void
+importDSLIS(documentDSLIS : JSON, cb : function(JSON, JSON)) : void

Figura 19: Diagramma della classe DocumentModel

## Descrizione

Classe che rappresenta la logica dell'applicazione nell'ambito dei Document, creati mediante DSLIS.

Utilizza il design pattern Module per dare visibilità o meno ai metodi.

## Utilizzo

Viene utilizzata per fornire le proprietà, gli ACL (Access Control Lists), le relazioni, i ruoli ed i metodi riguardanti l'ambito dei Document. La classe sarà relazionata con il database MongoDB dell'applicazione, rappresentato dalla classe Back-end::Datasources::DatabaseDatasource.

## Classi ereditate

- Back-end::Models::DSLISModels::DSLISModel

## Relazioni con altre classi

- Back-end::Models::DSLISModels::ConvertTo
- Back-end::Datasources::DatabaseDatasource
- Back-end::EmailDatasource
- Back-end::LoopBackStorageComponent

## Attributi

### **-documentData: JSON**

Rappresenta i dati di un Document dell'azienda.

- id di tipo *string* che rappresenta l'identificativo del Document;
- DSLIS di tipo *string* che rappresenta il codice DSL del Document.

### **conv: ConvertTo**

Rappresenta il riferimento alla classe ConvertTo per la conversione di un Document in un determinato formato.

## Metodi

### **+DocumentModel(conv: ConvertTo, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di ritornare un JS Object estraendo i dati da un Document.

#### **– conv: ConvertTo**

Rappresenta il riferimento alla classe ConvertTo per la conversione di un Document in un determinato formato;

#### **– cb: function(JSON, JS Object)**

Rappresenta la callback che il metodo deve chiamare al termine della creazione. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

### **+getDocuments(cb: function(JSON, JSON)) : void**

Questo metodo si occupa di ritornare la lista dei Document a cui l'utente ha accesso;

#### **– cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della richiesta. Nel caso di errori server verrà ritornato un JSON costruito appositamente,

rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente i dati di tutti i Model in caso di successo, mentre per l'esito negativo una stringa rappresentante l'errore.

**+createDocument(newDocument: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di creare un Document nell'Azienda;

– **newDocument: JSON**

Rappresenta i dati del Document da creare;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della creazione del nuovo Document. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+editDocument(id: string, document: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di modificare un Document dell'Azienda;

– **id: string**

Rappresenta l'identificativo del Document da modificare;

– **document: JSON**

Rappresenta i nuovi dati da rimpiazzare a quelli del Document;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della modifica del Document. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+deleteDocument(id: string, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di eliminare un Document dell'Azienda;

– **id: string**

Rappresenta l'identificativo del Document da eliminare;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della rimozione del Document. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+searchDocument(value: JSON, email: string, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di cercare un Document dell'Azienda;

– **value: string**

Rappresenta i valori da ricercare nei Document memorizzati;

– **email: string**

Rappresenta l'email dell'utente che effettua la ricerca;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della ricerca dei Document. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà

ritornato il secondo file JSON contenente i dati di tutti i Documents che matchano con la ricerca in caso di successo, mentre per l'esito negativo una stringa rappresentante l'errore.

**+executeDocument(id: string, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di eseguire un Document dell'Azienda;

– **id: string**

Rappresenta l'identificativo del Document da eseguire;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'esecuzione. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+executeDocument(collection: JSON, id: string, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di eseguire un Document appartenente ad una Collection dell'Azienda;

– **collection: JSON**

Rappresenta la Collection che contiene il Document da eseguire;

– **id: string**

Rappresenta l'identificativo del Document da eseguire;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'esecuzione. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+export(id: string, format: string, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di esportare i dati visualizzati da un Document dell'Azienda;

– **id: string**

Rappresenta l'identificativo del Document di cui esportare i dati;

– **format: string**

Rappresenta il formato di esportazione dei dati del Document;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'esportazione. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+retrieveDocument(id: string, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di recuperare il codice di un Document dell'Azienda;

– **id: string**

Rappresenta l'identificativo del Document da ritornare;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della richiesta.

Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+exportDSLIS(id: string, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di esportare il codice di un Document dell'Azienda;

**- id: string**

Rappresenta l'identificativo del Document di cui esportare il codice;

**- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'esportazione. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+importDSLIS(documentDSLIS: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di importare il codice di un Document dell'Azienda.

**- id: string**

Rappresenta l'identificativo del Document di cui importare il codice;

**- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'importazione. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**-extract(id: string, cb: function(JSON, JS Object))**

Questo metodo si occupa di ritornare un JS Object estraendo i dati da un Document.

**- id: string**

Rappresenta l'identificativo del Document;

**- cb: function(JSON, JS Object)**

Rappresenta la callback che il metodo deve chiamare al termine dell'estrazione. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

## DashboardModel

<b>DashboardModel</b>
-dashboardData : JSON
+getDashboards(cb : function(JSON, JSON)) : void
+createDashboard(newDashboard : JSON, cb : function(JSON, JSON)) : void
+editDashboard(id : string, dashboard : JSON, cb : function(JSON, JSON)) : void
+deleteDashboard(id : string, cb : function(JSON, JSON)) : void
+retrieveDashboard(id : string, cb : function(JSON, JSON)) : void
+searchDashboard(value : JSON, email : string, cb : function(JSON, JSON)) : void
+exportDashboard(id : string, cb : function(JSON, JSON)) : void
+exportDSLIS(id : string, cb : function(JSON, JSON)) : void
+importDSLIS(dashboardDSLIS : JSON, cb : function(JSON, JSON)) : void

Figura 20: Diagramma della classe DashboardModel

### Descrizione

Classe che rappresenta la logica dell'applicazione nell'ambito delle Dashboard, create mediante DSLIS.  
Utilizza il design pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per fornire le proprietà, gli ACL (Access Control Lists), le relazioni, i ruoli ed i metodi riguardanti l'ambito delle Dashboard. La classe sarà relazionata con il database MongoDB dell'applicazione, rappresentato dalla classe Back-end::Datasources::DatabaseDatasource.

### Classi ereditate

- Back-end::Models::DSLISModels::DSLISModel

### Relazioni con altre classi

- Back-end::Datasources::DatabaseDatasource
- Back-end::LoopBackStorageComponent

### Attributi

#### **dashboardData: JSON**

Rappresenta i dati di una Dashboard dell'azienda.

- id di tipo *string* che rappresenta l'identificativo della Dashboard.
- DSLIS di tipo *string* che rappresenta il codice DSL della Dashboard;

## Metodi

**+getDashboards(cb: function(JSON, JSON) : void**

Questo metodo si occupa di ritornare la lista delle Dashboard a cui l'utente ha accesso;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della richiesta. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente i dati di tutte le Dashboard di successo, mentre per l'esito negativo una stringa rappresentante l'errore.

**+createDashboard(newDashboard: JSON, cb: function(JSON, JSON) : void**

Questo metodo si occupa di creare una Dashboard nell'azienda;

– **newDashboard: JSON**

Rappresenta i dati della Dashboard da creare;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della creazione della Dashboard. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+editDashboard(id: string, dashboard: JSON, cb: function(JSON, JSON) : void**

Questo metodo si occupa di modificare una Dashboard dell'azienda;

– **id: string**

Rappresenta l'identificativo della Dashboard da modificare;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della modifica della Dashboard. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+deleteDashboard(id: string, cb: function(JSON, JSON) : void**

Questo metodo si occupa di eliminare una Dashboard dell'azienda;

– **id: string**

Rappresenta l'identificativo della Dashboard da eliminare;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'eliminazione della Dashboard. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+retrieveDashboard(id: string, cb: function(JSON, JSON) : void**

Questo metodo si occupa di recuperare il codice di una Dashboard dell'azienda;

– **id: string**

Rappresenta l'identificativo della Dashboard di cui ritornare il codice;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della richiesta

della Dashboard. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

```
+searchDashboard(value: JSON, email: string, cb: function(JSON, JSON)) : void
```

Questo metodo si occupa di cercare una Dashboard nell'azienda;

- **value: JSON**

Rappresenta i valori da ricercare nelle Dashboard dell'azienda;

- **email: string**

Rappresenta l'utente che effettua la ricerca;

- **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della ricerca delle Dashboard. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente i dati di tutte le Dashboard che matchano con la ricerca, mentre per l'esito negativo una stringa rappresentante l'errore.

```
+executeDashboard(id: string, cb: function(JSON, JSON)) : void
```

Questo metodo si occupa di eseguire una Dashboard dell'azienda;

- **id: string**

Rappresenta l'identificativo della Dashboard da eseguire;

- **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'esecuzione della Dashboard. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

```
+exportDSLIS(id: string, cb: function(JSON, JSON)) : void
```

Questo metodo si occupa di esportare il codice DSL di una Dashboard dell'azienda;

- **id: string**

Rappresenta l'identificativo della Dashboard di cui esportare il codice;

- **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'esportazione del codice della Dashboard. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

```
+importDSLIS(dashboardDSLIS: JSON, cb: function(JSON, JSON)) : void
```

Questo metodo si occupa di importare il codice DSL di una Dashboard dell'azienda;

- **id: string**

Rappresenta l'identificativo della Dashboard di cui importare il codice;

- **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'importazione del codice della Dashboard. Nel caso di errori server verrà ritornato un JSON

costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

### CellModel

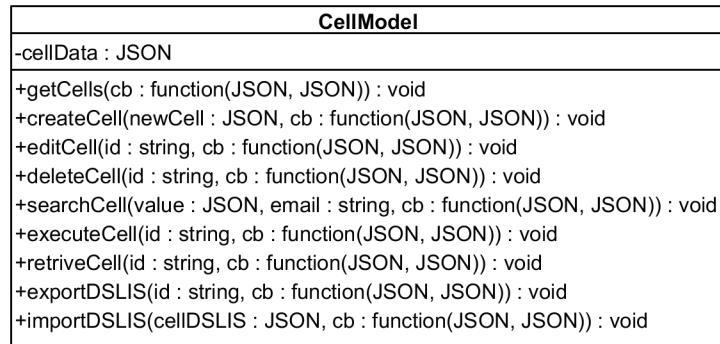


Figura 21: Diagramma della classe CellModel

### Descrizione

Classe che rappresenta la logica dell'applicazione nell'ambito delle Cell, create mediante DSLIS.

Utilizza il design pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per fornire le proprietà, gli ACL (Access Control Lists), le relazioni, i ruoli ed i metodi riguardanti l'ambito delle Cell. La classe sarà relazionata con il database MongoDB dell'applicazione, rappresentato dalla classe Back-end::Datasources::DatabaseDatasource.

### Classi ereditate

- Back-end::Models::DSLISModels::DSLISModel

### Relazioni con altre classi

- Back-end::Datasources::DatabaseDatasource
- Back-end::LoopBackStorageComponent

## Attributi

### **-cellData: JSON**

Rappresenta i dati di una Cell dell'azienda.

- id di tipo *string* che rappresenta l'identificativo della Cell;
- DSLIS di tipo *string* che rappresenta il codice DSL della Cell;

## Metodi

### **+getCells(cb: function(JSON, JSON)) : void**

Questo metodo si occupa di ritornare la lista delle Cell a cui l'utente ha accesso;

#### **- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della richiesta. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente i dati di tutte le Cell in caso di successo, mentre per l'esito negativo una stringa rappresentante l'errore.

### **+createCell(newCell: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di creare una Cell nell'azienda;

#### **- newCell: JSON**

Rappresenta i dati della Cell da creare;

#### **- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della creazione della Cell. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

### **+editCell(id: string, cell: JSON, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di modificare una Cell dell'azienda;

#### **- id: string**

Rappresenta l'identificativo della Cell da modificare;

#### **- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della modifica della Cell. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

### **+deleteCell(id: string, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di eliminare una Cell dell'azienda;

#### **- id: string**

Rappresenta l'identificativo della Cell da eliminare;

#### **- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'eliminazione della Cell. Nel caso di errori server verrà ritornato un JSON costruito

appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+searchCell(value: JSON, email: string, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di cercare una Cell nell'azienda;

– **value: JSON**

Rappresenta i valori da ricercare nelle Cell dell'azienda;

– **email: string**

Rappresenta l'utente che effettua la ricerca;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della ricerca delle Cell. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente i dati di tutte le Cell che matchano con la ricerca in caso di successo, mentre per l'esito negativo una stringa rappresentante l'errore.

**+executeCell(id: string, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di eseguire una Cell dell'azienda;

– **id: string**

Rappresenta l'identificativo della Cell da eseguire;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'esecuzione della Cell. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+retrieveCell(id: string, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di richiedere il codice di una Cell dell'azienda;

– **id: string**

Rappresenta l'identificativo della Cell richiesta;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine della richiesta della Cell. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

**+exportDSLIS(id: string, cb: function(JSON, JSON)) : void**

Questo metodo si occupa di esportare il codice DSL di una Cell dell'azienda;

– **id: string**

Rappresenta l'identificativo della Cell di cui esportare il codice;

– **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'esportazione del codice della Cell. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

```
+importDSLIS(cellDSLIS: JSON, cb: function(JSON, JSON)) : void
```

Questo metodo si occupa di importare il codice DSL di una Cell dell'azienda.

- **id: string**

Rappresenta l'identificativo della Cell di cui importare il codice;

- **cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'importazione del codice della Cell. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

## DSLInterpreterStrategy

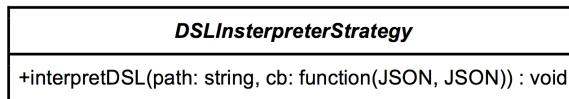


Figura 22: Diagramma della classe DSLInterpreterStrategy

**Descrizione** Classe astratta che definisce l'interfaccia dell'algoritmo di interpretazione del DSL utilizzato. E' il componente strategy del design pattern Strategy.

### Utilizzo

Viene utilizzata per incapsulare e rendere intercambiabile l'algoritmo di interpretazione del DSL. In questo modo, se in futuro vi fosse necessità di cambiare il DSL e di conseguenza anche il relativo algoritmo di interpretazione, basterà variare esso indipendentemente dal client che ne farà uso.

### Estensioni

- Back-end::Models::DSLISModels::DSLInterpreter

### Attributi

Assenti.

### Metodi

```
+interpretDSL(path: string, cb: function(JSON, JSON)) : void
```

Questo metodo si occupa di interpretare il contenuto del file DSL passatogli, per poi generare ed eseguire il codice derivato dalla trasformazione;

- **path: string**

Rappresenta la locazione del codice DSL da interpretare.

**- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'interpretazione. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

### DSLInterpreter

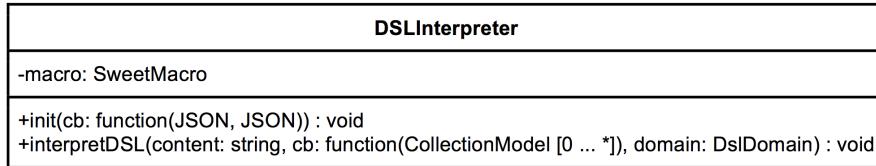


Figura 23: Diagramma della classe DSLInterpreter

**Descrizione** Classe che concretizza l'interprete del DSL. E' il componente ConcreteStrategy del design pattern Strategy. Viene utilizzato il componente sweet.js per creare delle macro rappresentanti le parole chiave del DSL.

**Utilizzo** Viene utilizzata per implementare l'algoritmo utilizzato nell'interfaccia Back-end::Models::DSLISModels::DSLInterpreterStrategy per l'interpretazione del linguaggio DSL. Conterrà al suo interno un metodo utile a generare il parser, a partire da una grammatica regolare.

#### Classi ereditate

- Back-end::Models::DSLISModels::DSLInterpreterStrategy

#### Attributi

**-macro: SweetMacro**

Rappresenta i dati di una macro di interpretazione del file DSL, utilizzata da sweet.js.

#### Metodi

**+init(cb: function(JSON, JSON)) : void**

Questo metodo viene invocato per impostare l'interprete nel modo corretto. Viene settato il modulo sweet.js a partire dal codice di definizione delle macro;

**- cb: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'inizializzazione. Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

```
+interpretDSL(content: string, cb: function(CollectionModel [0 ... *], JSON)
: void
```

Questo metodo si occupa di interpretare il contenuto del file DSL passatogli, per poi generare ed eseguire il codice derivato dalla trasformazione tramite le macro di sweet.js;

- **content: string**

Questo parametro rappresenta il contenuto testuale del file DSL da interpretare.

- **cb: function(CollectionModel [0 ... \*]), JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'interpretazione.

Nel caso di errori server verrà ritornato un JSON costruito appositamente, rappresentato dal primo parametro. Se non ci sono errori verrà ritornato il secondo file JSON contenente il successo o l'insuccesso dell'operazione.

#### 5.1.4 Back-end::Datasources

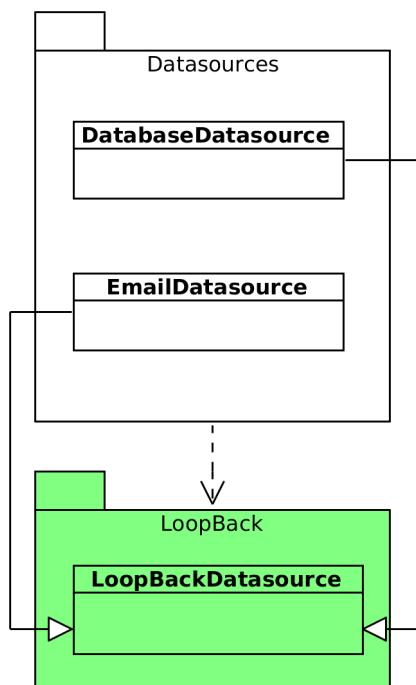


Figura 24: Diagramma delle classi del package Datasources

##### 5.1.4.1 Informazioni sul package

###### Descrizione

Package contenente le classi che rappresentano i servizi di back-end, come per esempio i database.

#### 5.1.4.2 Classi

##### DatabaseDatasource

###### Descrizione

Classe che rappresenta il database MongoDB dell'applicazione.  
Viene derivata dalla classe LoopBackDatasource del framework LoopBack.

###### Utilizzo

Viene utilizzata per fornire i metodi per connettersi al database dell'applicazione ed impostare le opzioni di connessione. Rappresenta il supporto di memorizzazione dei dati delle classi dichiarate nel package Back-end::Models. La classe utilizza i metodi offerti dalla classe Back-end::Connectors::MongoDBConnector, per gestire le richieste delle classi presenti nel package Back-end::Models riguardanti i dati del database di MaaS.

###### Classi ereditate

- LoopBack::LoopBackDatasource

###### Relazioni con altre classi

- Back-end::Connectors::MongoDBConnector

###### Attributi

Assenti.

###### Metodi

Assenti, in quanto utilizza i metodi forniti dal framework LoopBack.

##### EmailDatasource

###### Descrizione

Classe che rappresenta il servizio email associato all'applicazione.  
Viene derivata dalla classe LoopBackDatasource del framework LoopBack.

###### Utilizzo

Viene utilizzata per fornire i metodi per connettersi al servizio email dell'applicazione ed impostare le opzioni di connessione. Inoltre, fornisce il metodo per inviare un'email. Rappresenta il supporto per l'invio di Email, utile alle classi contenute nel package Back-end::Models.

### Classi ereditate

- LoopBack::LoopBackDatasource

### Relazioni con altre classi

- Back-end::Connectors::EmailConnector

### Attributi

Assenti.

### Metodi

Assenti, in quanto utilizza i metodi forniti dal framework LoopBack.

#### 5.1.5 Back-end::Connectors

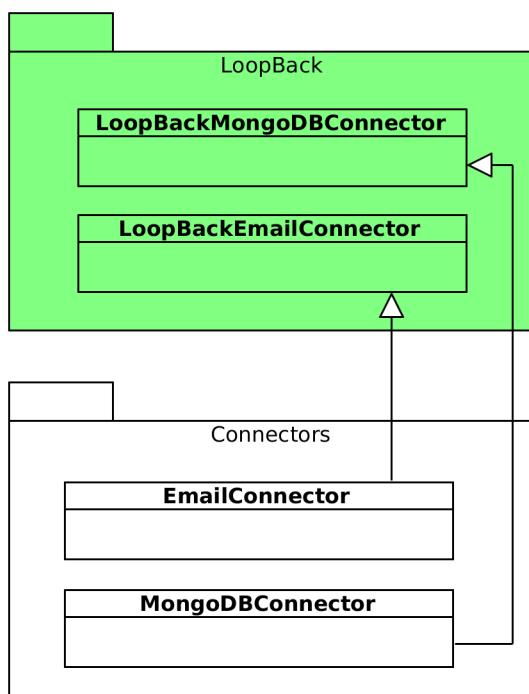


Figura 25: Diagramma delle classi del package Connectors

##### 5.1.5.1 Informazioni sul package

#### Descrizione

Package contenente le classi derivate dal framework LoopBack, che rappresentano i connettori alle risorse ed ai servizi di back-end e si occupano di comunicarci direttamente.

### Framework esterni

- LoopBack

#### 5.1.5.2 Classi

##### MongoDBConnector

###### Descrizione

Classe che contiene i metodi per interfacciarsi e gestire il database MongoDB dell'applicazione. Essa viene derivata dal connettore MongoDB offerto dal framework LoopBack.

###### Utilizzo

Viene utilizzata per fornire i metodi utili ad eseguire le operazioni sul database MongoDB dell'applicazione. Essa utilizza il metodo di connessione, con i relativi parametri, della classe DatabaseDatasource per connettersi al database.

###### Classi ereditate

- LoopBack::LoopBackMongoDBConnector

###### Attributi

Assenti.

###### Metodi

Assenti, in quanto utilizza i metodi forniti dal framework LoopBack.

##### EmailConnector

###### Descrizione

Classe che contiene i metodi per interfacciarsi e gestire il servizio di email dell'applicazione. Essa viene derivata dal connettore Email offerto dal framework LoopBack.

###### Utilizzo

Viene utilizzata per fornire i metodi utili ad eseguire le operazioni sull'email dell'applicazione. Essa utilizza il metodo di connessione, con i relativi parametri, della classe EmailDatasource per connettersi al servizio email.

## Attributi

Assenti.

## Metodi

Assenti, in quanto utilizza i metodi forniti dal framework LoopBack.

## Classi ereditate

- LoopBack::LoopBackEmailConnector

### 5.1.6 Back-end::Middlewares

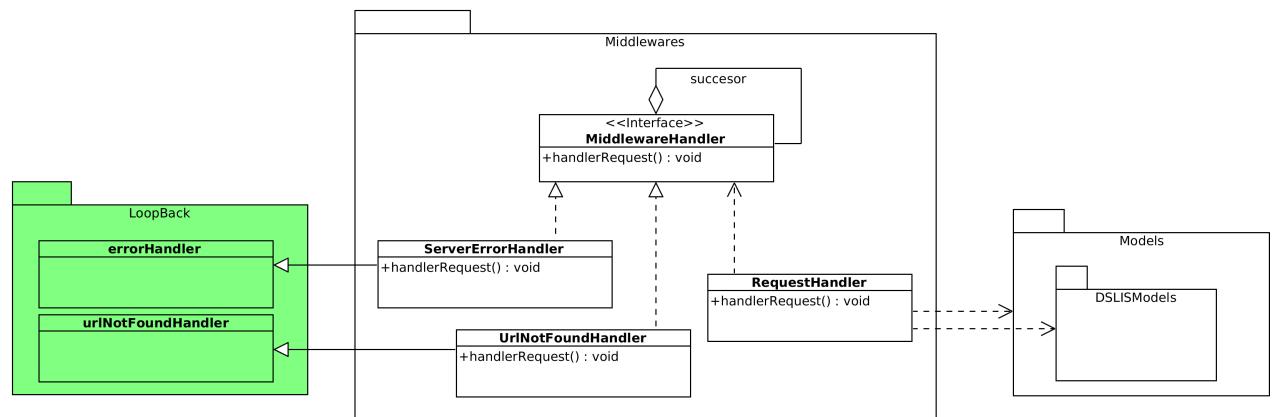


Figura 26: Diagramma delle classi del package Middlewares

#### 5.1.6.1 Informazioni sul package

##### Descrizione

Package contenente le classi che costituiscono gli handler della catena di chiamate utili a verificare la presenza di errori nelle richieste del client.

Viene utilizzato il design pattern Module per dare visibilità o meno ai metodi.

##### Framework esterni

- LoopBack

#### 5.1.6.2 Classi

##### MiddlewareHandler



Figura 27: Diagramma della classe MiddlewareHandler

### Descrizione

Classe che rappresenta la componente Handler del design pattern Chain of Responsibility.

### Utilizzo

Viene utilizzata per fornire l'interfaccia per gestire le richieste.

### Relazioni con altre classi

- Back-end::Middlewares::MiddlewareHandler

### Attributi

Assenti.

### Metodi

```
handlerRequest(req: request, res: response, next: function(JSON, JSON)) : void
```

Questo metodo si occupa di gestire la richiesta in ingresso al server, in particolare è il primo elemento della catena delle responsabilità che passerà il controllo ai successivi;

– **req: request**

Questo oggetto rappresenta la richiesta di tipo request arrivata al server che il metodo deve gestire;

– **res: response**

Questo oggetto rappresenta la risposta che il server dovrà dare al termine dell'elaborazione;

– **next: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'elaborazione della richiesta per passare il controllo ai successivi middleware. Se non ci sono errori verrà ritornato un file JSON contenente il successo o l'insuccesso dell'operazione. Nel caso di errori server verrà ritornato un JSON e verrà attivata la catena di gestione delle richieste.

### UrlNotFoundHandler



Figura 28: Diagramma della classe UrlNotFoundHandler

### Descrizione

Classe che rappresenta la componente ConcreteHandler del design pattern Chain of Responsibility. Essa viene derivata dalla classe middleware offerta dal framework LoopBack Back-end::Middlewares::LoopBackurlNotFoundHandler. Utilizza il design pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per gestire le richieste del client e verificare la presenza di errori riguardanti gli URL errati in esse.

### Classi ereditate

- LoopBack::LoopBackurlNotFoundHandler

### Relazioni con altre classi

- Back-end::Middlewares::MiddlewareHandler

### Metodi

```
handlerRequest(req: request, res: response, next: function(JSON, JSON)) : void
```

Questo metodo si occupa di gestire la richiesta in ingresso al server, in particolare è l'elemento della catena delle responsabilità che controlla che l'URL inserito dall'utente sia corretto;

– **req: request**

Questo oggetto rappresenta la richiesta di tipo request arrivata al server che il metodo deve gestire;

– **res: response**

Questo oggetto rappresenta la risposta che il server dovrà dare al termine dell'elaborazione;

– **next: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'elaborazione della richiesta per passare il controllo ai successivi middleware. Se non ci sono errori verrà ritornato un file JSON contente il successo o l'insuccesso dell'operazione. Nel caso di errori server verrà ritornato un JSON e verrà attivata la catena di gestione delle richieste.

## ServerErrorHandler

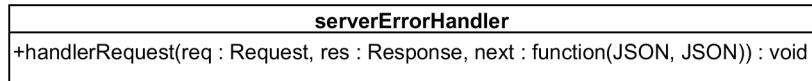


Figura 29: Diagramma della classe ServerErrorHandler

### Descrizione

Classe che rappresenta la componente ConcreteHandler del design pattern Chain of Responsibility. Essa viene derivata dalla classe middleware offerta dal framework LoopBack Back-end::Middlewares::LoopBackErrorHandler.

Utilizza il design pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per gestire le richieste del client e verificare la presenza di errori riguardanti l'impossibilità del server nel gestirle.

### Classi ereditate

- LoopBack::LoopBackErrorHandler

### Relazioni con altre classi

- Back-end::Middlewares::MiddlewareHandler

### Attributi

Assenti.

### Metodi

```
handlerRequest(req: request, res: response, next: function(JSON, JSON)) : void
```

Questo metodo si occupa di gestire la richiesta in ingresso al server, in particolare è l'elemento della catena delle responsabilità che controlla l'eventuale presenza di errori lato server;

– **req: request**

Questo oggetto rappresenta la richiesta di tipo request arrivata al server che il metodo deve gestire;

– **res: response**

Questo oggetto rappresenta la risposta che il server dovrà dare al termine dell'elaborazione;

– **next: function(JSON, JSON)**

Rappresenta la callback che il metodo deve chiamare al termine dell'elaborazione della richiesta per passare il controllo ai successivi middleware. Se non ci sono errori verrà ritornato un file JSON contenente il successo o l'insuccesso dell'operazione. Nel caso di errori server verrà ritornato un JSON e verrà attivata la catena di gestione delle richieste.

### RequestHandler



Figura 30: Diagramma della classe RequestHandler

### Descrizione

Classe che rappresenta la componente ConcreteHandler del design pattern Chain of Responsibility.

### Utilizzo

Viene utilizzata per terminare la catena di middleware nel caso in cui nessuno di essi trovi degli errori, in modo da passare la richiesta del client alle REST API delle classi contenute nel package Back-end::Models.

### Relazioni con altre classi

- Back-end::Models::UserModel
- Back-end::Models::SuperAmministratoreModel
- Back-end::Models::CompanyModel
- Back-end::Models::ExternalDatabaseModel
- Back-end::Models::DSLISModels::DocumentModel
- Back-end::Models::DSLISModels::CollectionModel
- Back-end::Models::DSLISModels::DashboardModel
- Back-end::Models::DSLISModels::CellModel

### Attributi

Assenti.

## Metodi

```
handlerRequest(req: request, res: response, next: function(JSON, JSON)) : void
```

Questo metodo si occupa di gestire la richiesta in ingresso al server, in particolare è l'elemento finale della catena delle responsabilità che inoltra la richiesta alla componente Model visto che non si sono verificati errori;

- `req: request`

Questo oggetto rappresenta la richiesta di tipo `request` arrivata al server che il metodo deve gestire;

- `res: response`

Questo oggetto rappresenta la risposta che il server dovrà dare al termine dell'elaborazione;

- `next: function(JSON, JSON)`

Rappresenta la callback che il metodo deve chiamare al termine dell'elaborazione della richiesta per passare il controllo ai successivi middleware. Se non ci sono errori verrà ritornato un file `JSON` contenente il successo o l'insuccesso dell'operazione. Nel caso di errori server verrà ritornato un `JSON` e verrà attivata la catena di gestione delle richieste.

## 5.2 Interfaccia REST

### 5.2.1 Comunicazione tra client e server

Come visto in precedenza, per l'implementazione del *back-end<sub>G</sub>* di *MaaS<sub>G</sub>* si è deciso di utilizzare il framework *LoopBack<sub>G</sub>* che permette in modo semplice e veloce di creare una solida interfaccia *REST<sub>G</sub>*, generando automaticamente un insieme di *API<sub>G</sub>* per ogni modello.

#### 5.2.1.1 Oggetti JSON

Ad ogni API il server risponde con un oggetto in formato `JSON` per fornire le informazioni.

**Errori** Nell'eventualità fornisce invece un messaggio di errore, inviato con un pacchetto HTTP di tipo 4xx (errore nella richiesta del client) o 5xx (errore nella risposta del server), nel seguente formato `JSON`:

```
{  
    "code": [codice numerico dell'errore],  
    "message": [descrizione testuale dell'errore],  
    "data": [eventuali dati aggiuntivi sull'errore]  
}
```

Di seguito sono mostrati i due tipi d'errore trattati nel formato `JSON`:

- **Errore nella richiesta:**

```
{
    "code": 400,
    "message": "Object not found"
}
```

- Errore del server:

```
{
    "code": 500,
    "message": "Server error"
}
```

**Input al server** Il formato JSON è adottato anche per la transizione dei dati in input al server, per elaborare le richieste, la struttura è la seguente:

```
{
    "request": {
        "method": [GET / POST / PUT / DELETE],
        "path": [percorso della richiesta tramite API]
    },
    "data": [eventuali dati in input]
}
```

Un esempio di richiesta può essere quello di "GET /User/email", dove viene inviata l'email di un utente di cui si vogliono ricevere i dati:

```
{
    "request": {
        "method": "GET",
        "path": "/User"
    },
    "data": "mariorossi@gmail.com"
}
```

**Output del server** Oltre a gestire output d'errore il formato JSON è utilizzato anche per ritornare i risultati di interrogazioni ai datasource di LoopBack. Il formato utilizzato è il seguente:

```
{
    "success": [rappresenta l'esito dell'operazione]
    "response": [rappresenta il risultato della query, in
caso di insuccesso e' null]
}
```

Di seguito è mostrato l'esempio di risultato della richiesta "GET /User/email" vista prima, nel caso di successo e di insuccesso:

- **Richiesta GET User**(andata a buon fine):

```
{
    "success": "true",
    "response": {
        "email": "mariorossi@gmail.com",
        "name": "Mario",
        "surname": "Rossi",
        "date": "1/1/2000",
        "gender": "male"
    }
}
```

- **Richiesta Get User**(non andata a buon fine):

```
{
    "success": "false",
    "response": "null"
}
```

Di seguito sono descritte le API del back-end, elencando per ogni risorsa REST il tipo di metodo che è possibile richiedere su di essa e i permessi richiesti per poter effettuare la richiesta.

### 5.2.2 Permessi

I tipi di permessi possibili sono:

- **Utente Non Autenticato:** questa risorsa può essere richiesta solo dagli utenti autenticati a MaaS;
- **Utente Autenticato:** questa risorsa può essere richiesta solo dagli utenti autenticati a MaaS, con qualsiasi ruolo;
- **Ospite:** tale risorsa può essere richiesta solo da utenti con il ruolo di Ospite.
- **Membro:** tale risorsa può essere richiesta solo da utenti con il ruolo di Membro.
- **Amministratore:** tale risorsa può essere richiesta solo da utenti con il ruolo di Amministratore.
- **Proprietario:** tale risorsa può essere richiesta solo da utenti con il ruolo di Proprietario.
- **Super Amministratore:** tale risorsa può essere richiesta solo da utenti con il ruolo di Super Amministratore.

### 5.2.3 Richieste HTTP

#### 5.2.3.1 Richieste Users

##### Richiesta POST /Users

- **Descrizione:** Crea una richiesta di registrazione.
- **Richiesta HTTP:** /Users
- **Metodo:** POST
- **Permessi:** Utente non autenticato, Utente invitato.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa /Users. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta all'UserModel che chiama il metodo createUser() il quale si occupa di registrare un utente. L'UserModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

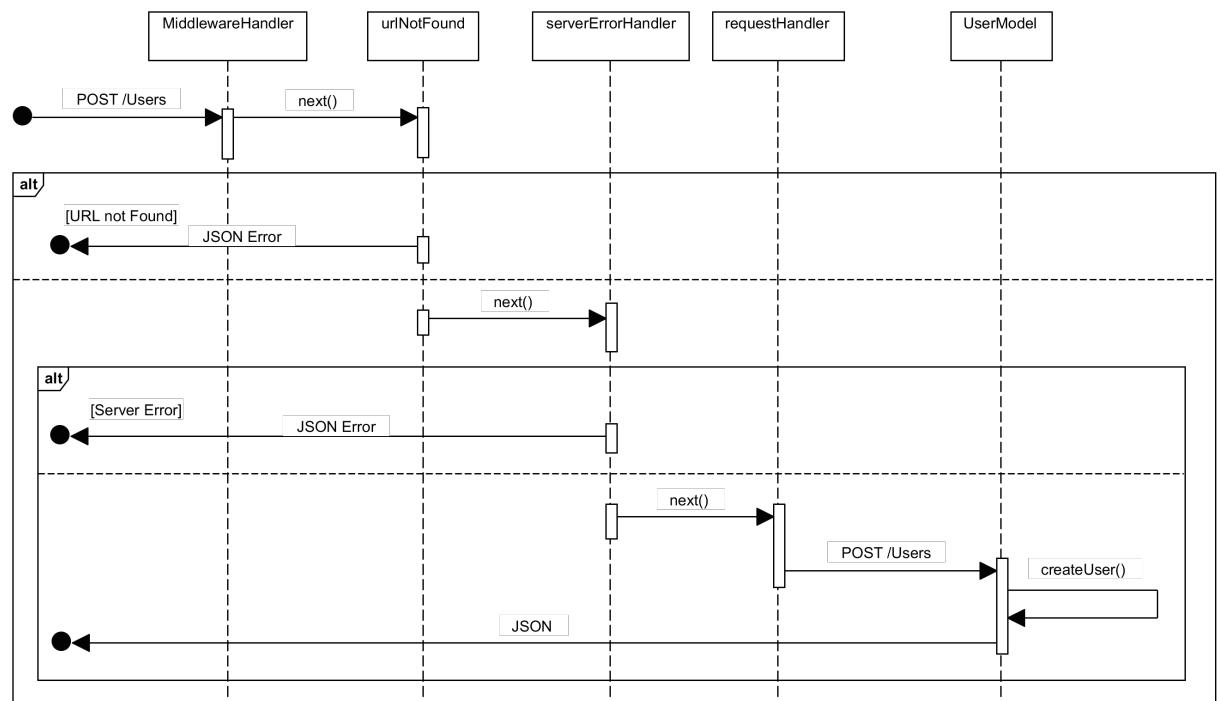


Figura 31: Diagramma di sequenza: POST /User

##### Richiesta POST /Users/login

- **Descrizione:** Esegue l'autenticazione e crea la sessione corrispondente all'utente.
- **Richiesta HTTP:** /Users/login
- **Permessi:** Utente non autenticato.
- **Metodo:** POST
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa /Users/login. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta all'UserModel che chiama il metodo loginUser() il quale si occupa di registrare un utente. L'UserModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

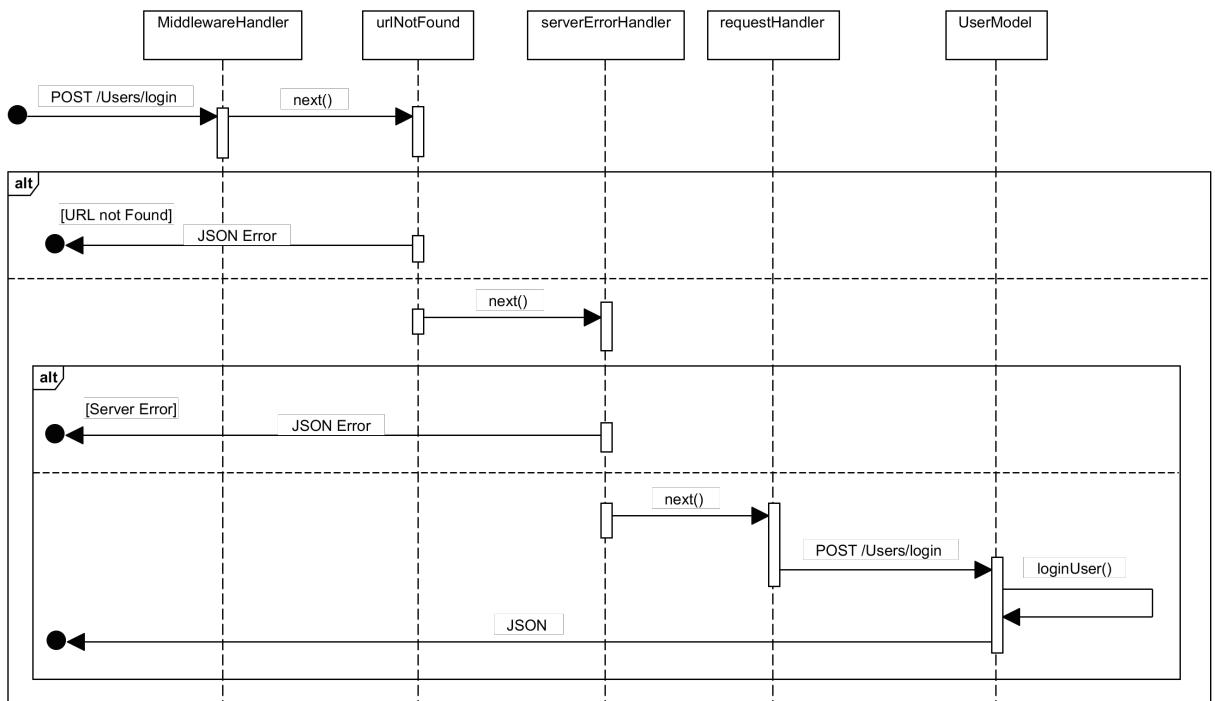


Figura 32: Diagramma di sequenza: POST /Users/login

### Richiesta PUT /Users

- **Descrizione:** Modifica i dati dell'utente.
- **Richiesta HTTP:** /Users
- **Metodo:** PUT
- **Permessi:** Utente autenticato, Super Amministratore.

- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta PUT per la risorsa /Users. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta all'UserModel che chiama il metodo editUser() il quale si occupa di modificare i dati di un utente. L'UserModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

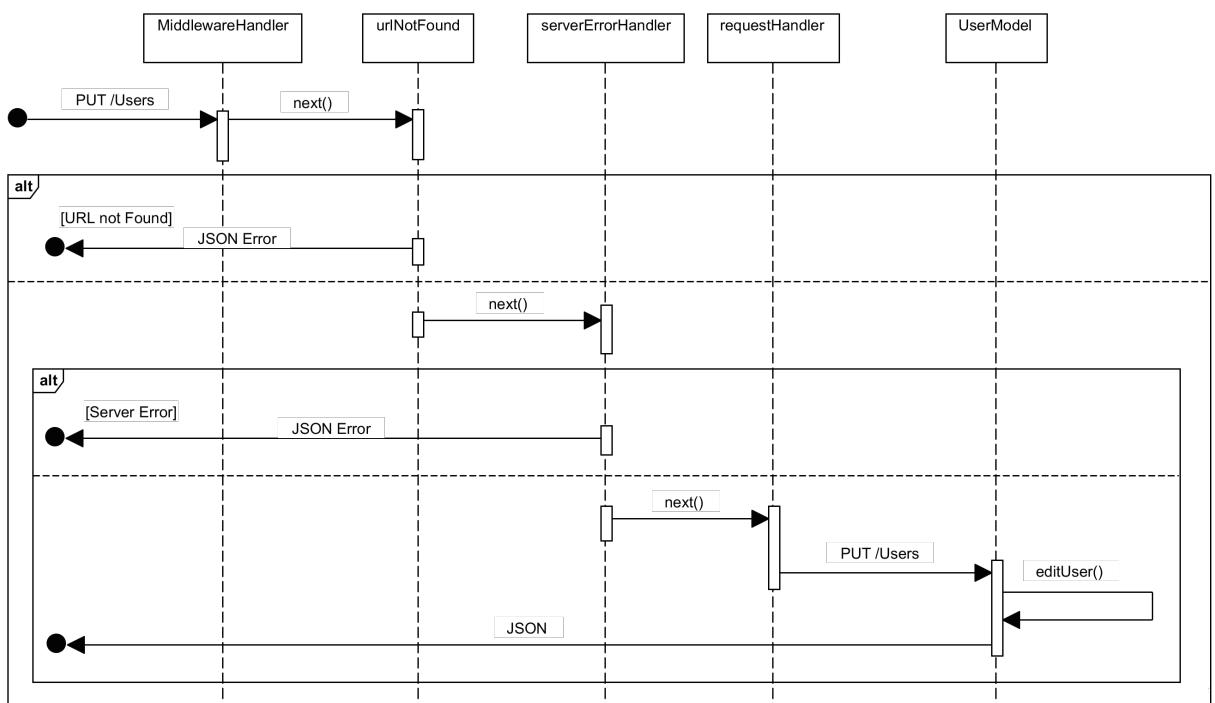


Figura 33: Diagramma di sequenza: PUT /Users

#### Richiesta POST /Users/logout

- Descrizione:** Elimina la sessione utente, corrispondente al logout.
- Richiesta HTTP:** /Users/logout
- Metodo:** POST
- Permessi:** Utente autenticato.
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa /Users/logout. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta all'UserModel

che chiama il metodo `logoutUser()` il quale si occupa di effettuare il logout di un utente distruggendo la sua sessione. L'UserModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

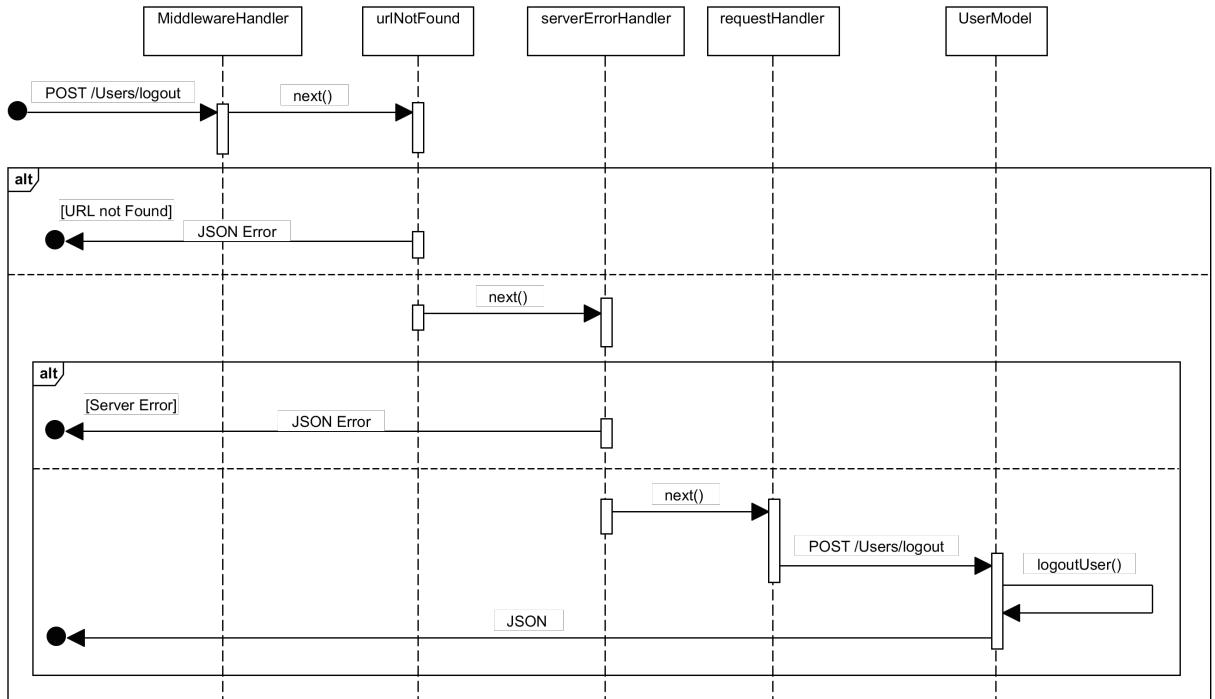


Figura 34: Diagramma di sequenza: POST /Users/logout

### Richiesta GET /Users

- **Descrizione:** Restituisce i dati relativi agli utenti.
- **Richiesta HTTP:** /Users
- **Metodo:** GET
- **Permessi:** Proprietario, Amministratore, Super Amministratore.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Users. La richiesta viene passata al `MiddlewareHandler` il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta all'`UserModel` che chiama il metodo `getUsers()` il quale richiede all'`UserModel` i dati relativi ad ogni utente. L'`UserModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

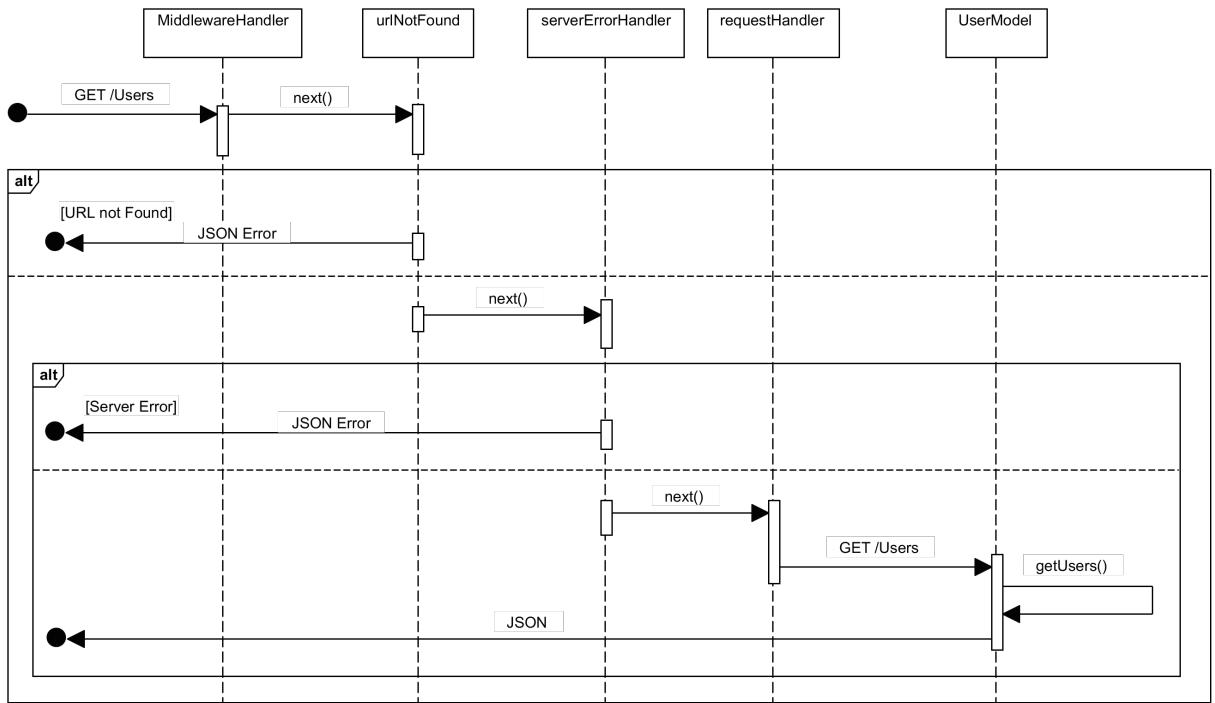


Figura 35: Diagramma di sequenza: GET /Users

### Richiesta GET /Users/{email}

- **Descrizione:** Restituisce i dati di un utente.
- **Richiesta HTTP:** /Users/{email}
- **Metodo:** GET
- **Permessi:** Utente autenticato.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Users/{email}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta all'UserModel che chiama il metodo getUser(email) il quale richiede i dati di un utente specifico. L'UserModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

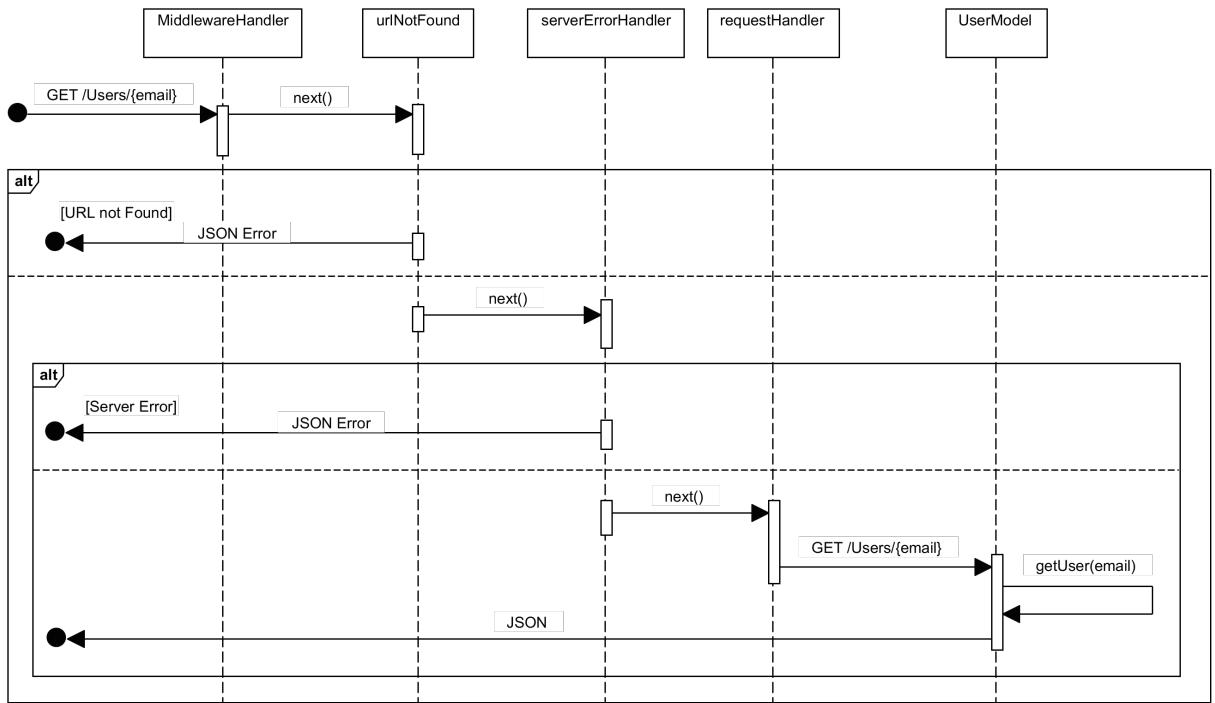


Figura 36: Diagramma di sequenza: GET /Users/{email}

#### Richiesta DELETE /Users/deleteUser/{email}

- **Descrizione:** Elimina un utente.
- **Richiesta HTTP:** /Users/deleteUser/{email}
- **Metodo:** DELETE
- **Permessi:** Proprietario, Amministratore, Super Amministratore.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta DELETE per la risorsa /Users/deleteUser/{email}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta all'UserModel che chiama il metodo deleteUser(email) il quale si occupa di eliminare un utente. L'UserModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

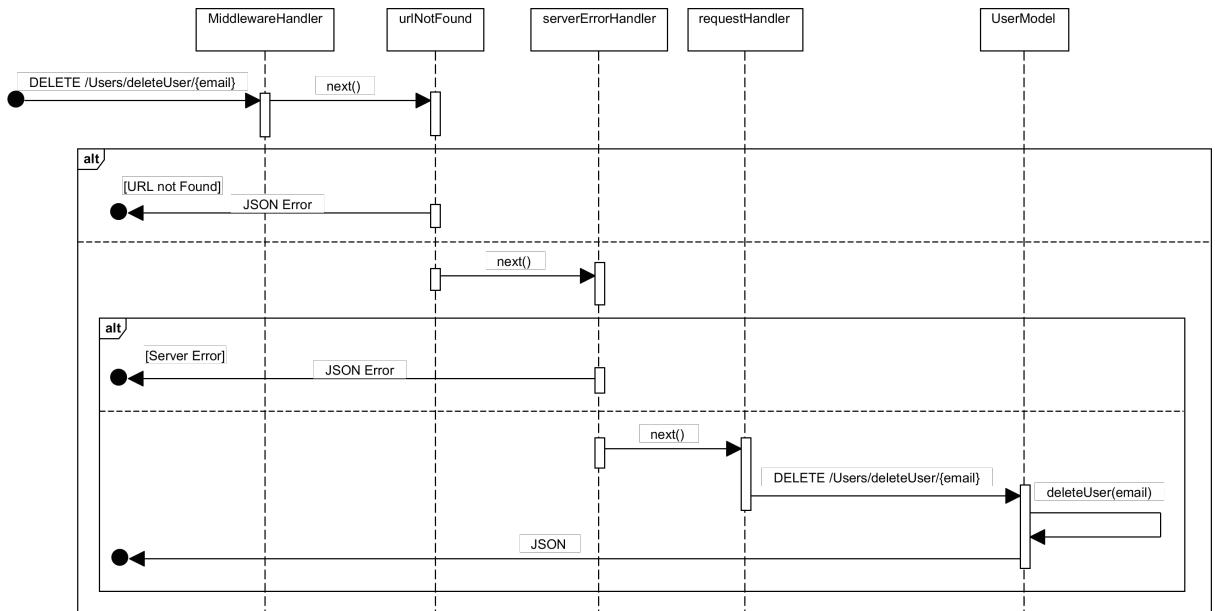


Figura 37: Diagramma di sequenza: `DELETE /Users/deleteUser/{email}`

#### Richiesta GET /User/SearchUser/{value}

- **Descrizione:** Dato un valore, ricerca un utente.
- **Richiesta HTTP:** `/User/SearchUser/{value}`
- **Metodo:** GET
- **Permessi:** Super Amministratore
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa `/User/SearchUser/{value}`. La richiesta viene passata al `MiddlewareHandler` il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta all'`UserModel` che chiama il metodo `searchUser(value)` che dato un parametro identificativo ritorna gli utenti interessati. L'`UserModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

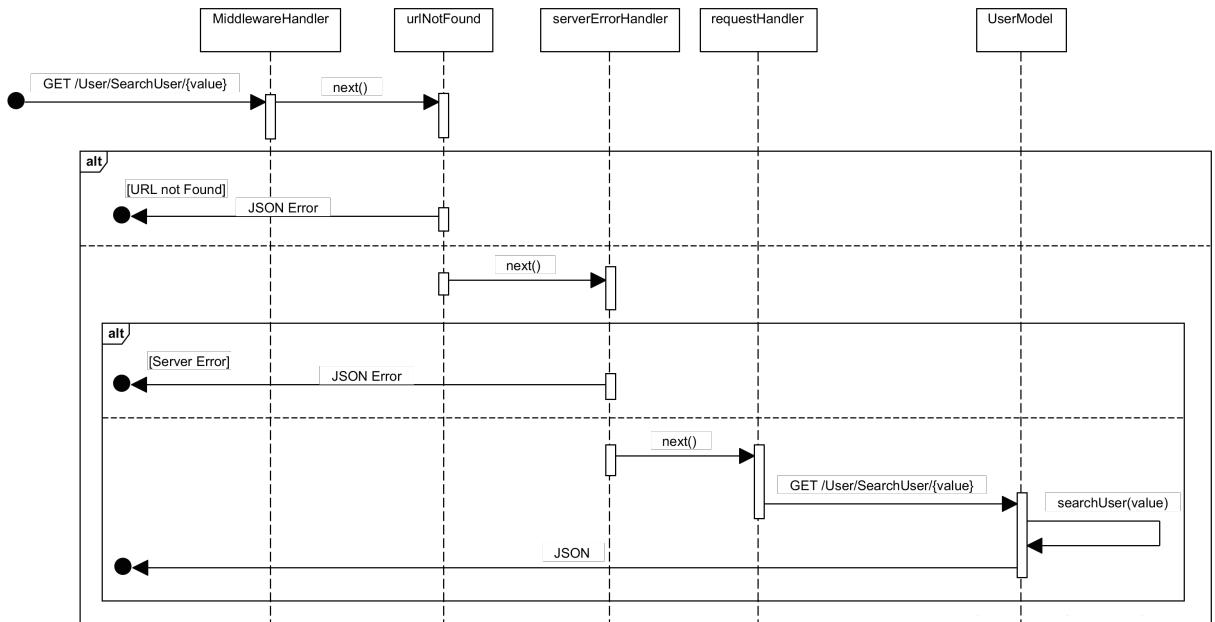


Figura 38: Diagramma di sequenza: `GET /User/SearchUser/{value}`

#### Richiesta POST /Users/forgotPassword

- **Descrizione:** Richiede il recupero della password.
- **Richiesta HTTP:** `/Users/forgotPassword`
- **Metodo:** POST
- **Permessi:** Utente non autenticato.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa `/Users/forgotPassword`. La richiesta viene passata al `MiddlewareHandler` il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta all'`UserModel` che chiama il metodo `forgotPassword()` che permette all'utente di recuperare la password dimenticata. L'`UserModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

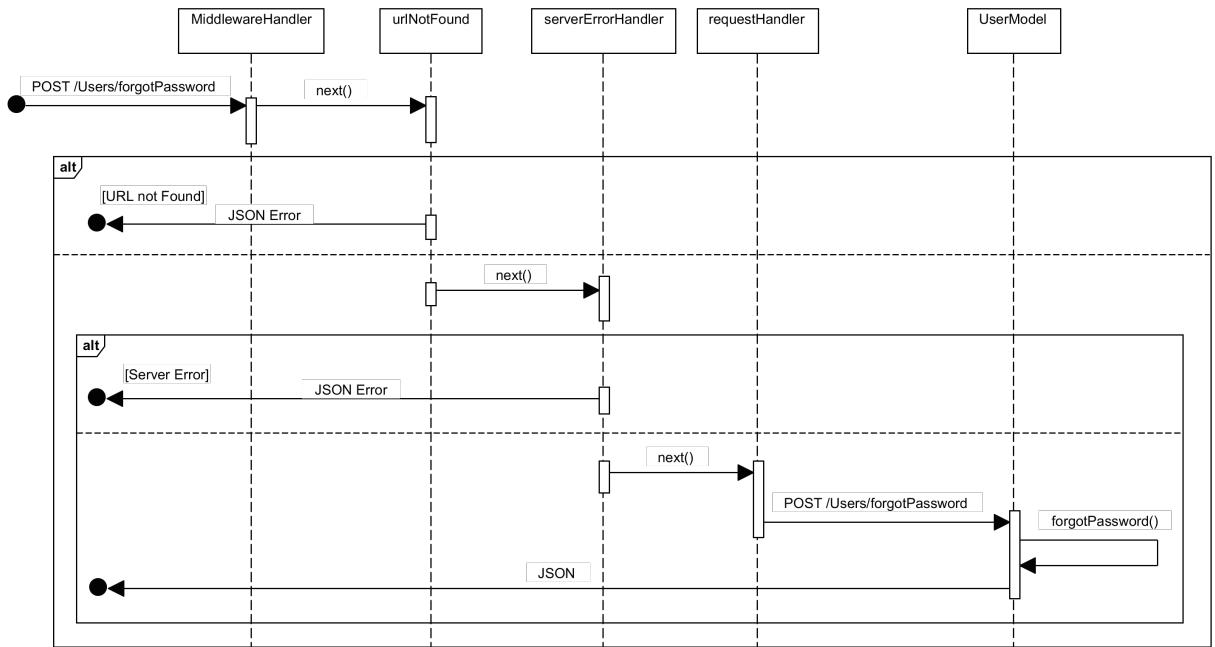
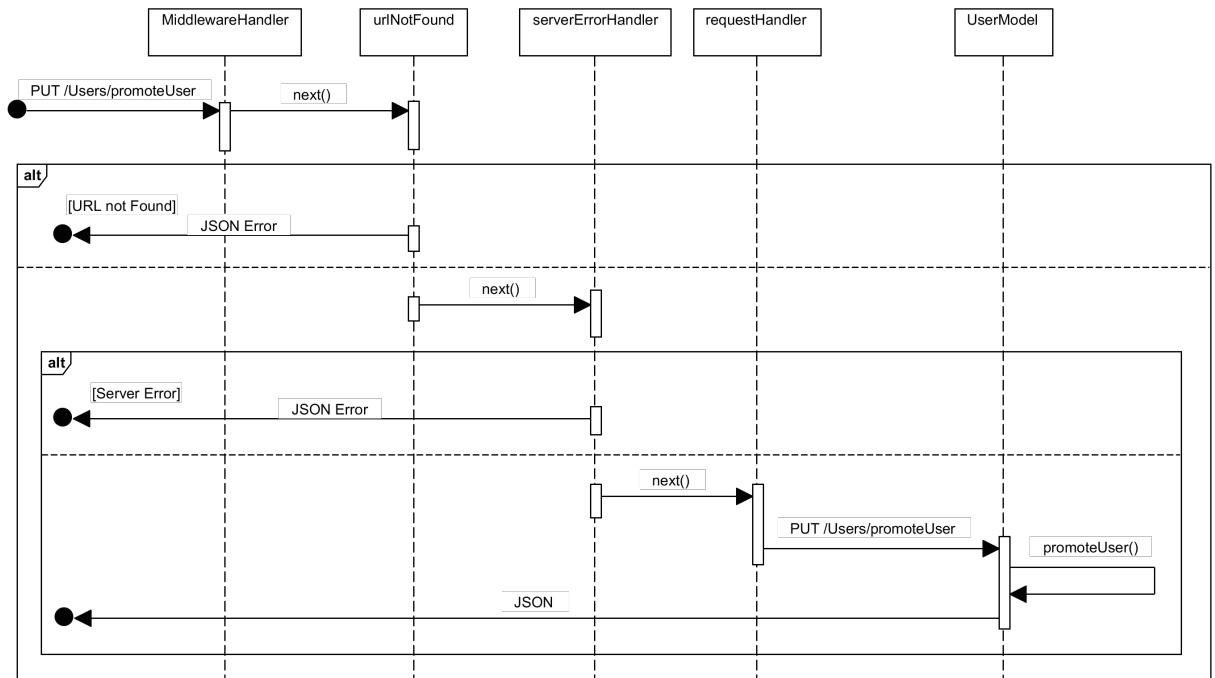


Figura 39: Diagramma di sequenza: POST /Users/forgotPassword

#### Richiesta PUT /Users/changeRole

- **Descrizione:** Richiesta di promozione del ruolo di un utente.
- **Richiesta HTTP:** /Users/changeRole
- **Metodo:** PUT
- **Permessi:** Proprietario, Amministratore.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta PUT per la risorsa /Users/changeRole. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta all'UserModel che chiama il metodo changeRole() che permette di cambiare il ruolo di un utente. L'UserModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.


 Figura 40: Diagramma di sequenza: `PUT /Users/changeRole`

#### Richiesta POST /Users/sendInvite

- **Descrizione:** Invio di una richiesta di registrazione all'azienda.
- **Richiesta HTTP:** `/Users/sendInvite`
- **Metodo:** POST
- **Permessi:** Proprietario, Amministratore.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa `/Users/sendInvite`. La richiesta viene passata al `MiddlewareHandler` il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta all'`UserModel` che chiama il metodo `sendInvite()` che permette di invitare un utente nell'azienda. L'`UserModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

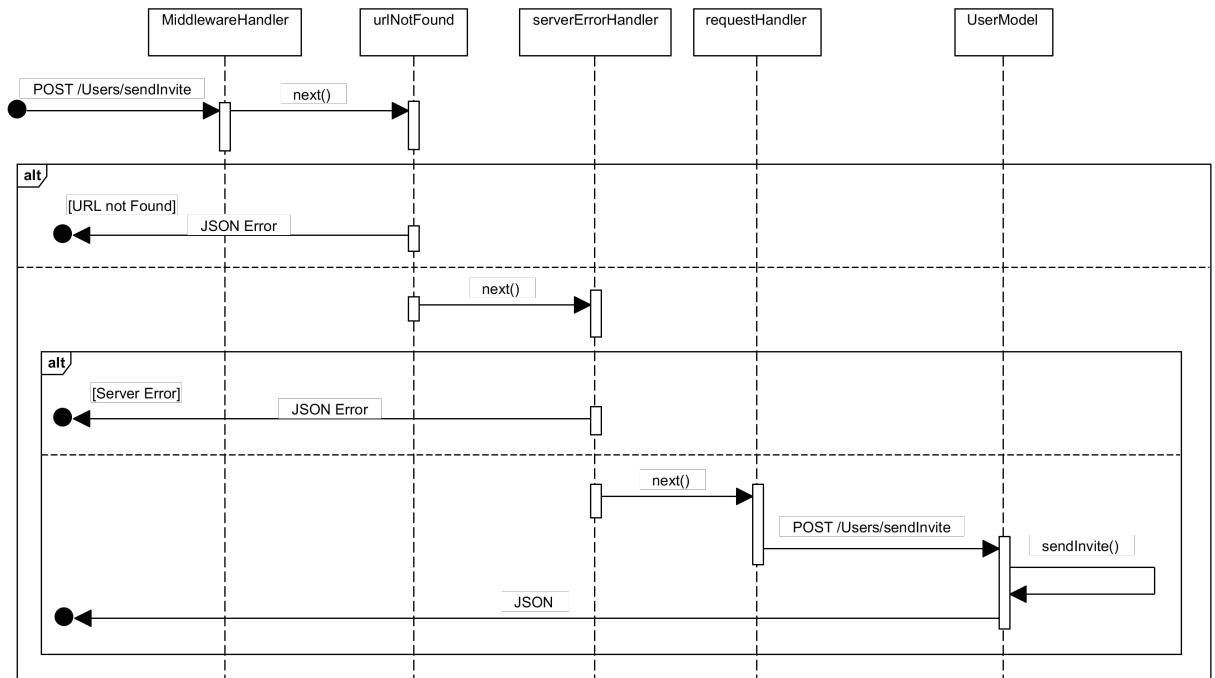


Figura 41: Diagramma di sequenza: POST /Users/sendInvite

### 5.2.3.2 Richieste Super Admin

#### Richiesta POST /SuperAdmin/login

- Descrizione:** Esegue l'autenticazione e crea la sessione corrispondente al Super Amministratore.
- Richiesta HTTP:** /SuperAdmin/login
- Metodo:** POST
- Permessi:** Super Amministratore.
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa /SuperAdmin/login. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al SuperAmministratoreModel che chiama il metodo loginSuperAdmin() che effettua il login di un Super Amministratore. Il SuperAmministratoreModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

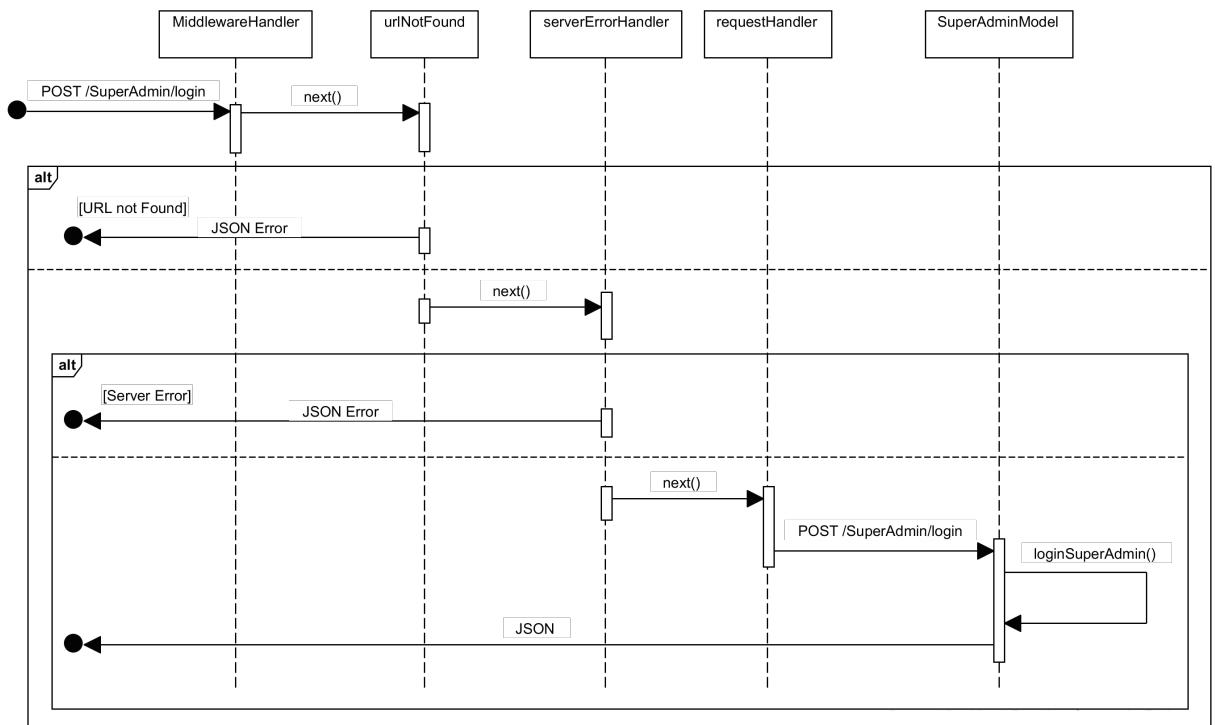


Figura 42: Diagramma di sequenza: POST /SuperAdmin/login

#### Richiesta POST /SuperAdmin/logout

- **Descrizione:** Elimina la sessione del Super Amministratore, corrispondente al logout.
- **Richiesta HTTP:** /SuperAdmin/logout
- **Metodo:** POST
- **Permessi:** Super Amministratore.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa /SuperAdmin/logout. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al SuperAmministratoreModel che chiama il metodo logoutSuperAdmin() che effettua il logout di un Super Amministratore. Il SuperAmministratoreModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

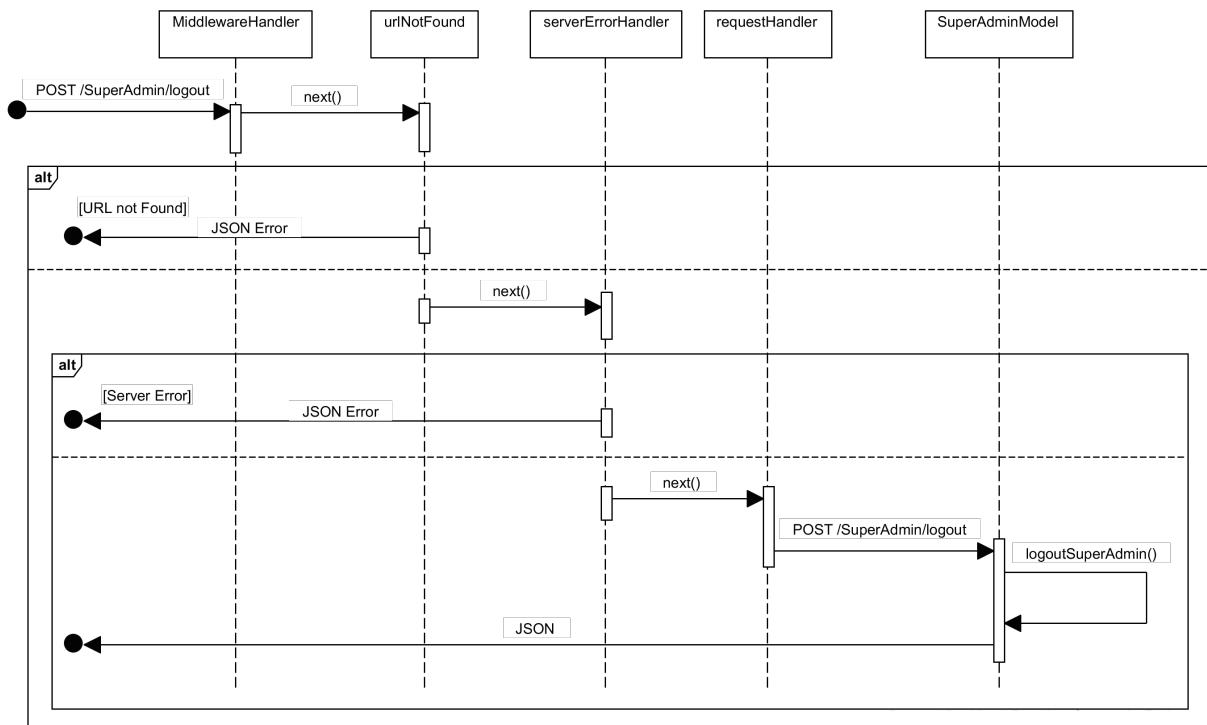


Figura 43: Diagramma di sequenza: POST /SuperAdmin/logout

#### Richiesta POST /SuperAdmin/impersonateUser

- **Descrizione:** Consente l'impersonificazione del Super Amministratore in un utente.
- **Richiesta HTTP:** POST
- **Metodo:** /SuperAdmin/impersonateUser
- **Permessi:** Super Amministratore.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa /SuperAdmin/impersonateUser. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al SuperAmministratoreModel che chiama il metodo impersonateUser() che permette l'impersonificazione di un utente. Il SuperAmministratoreModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

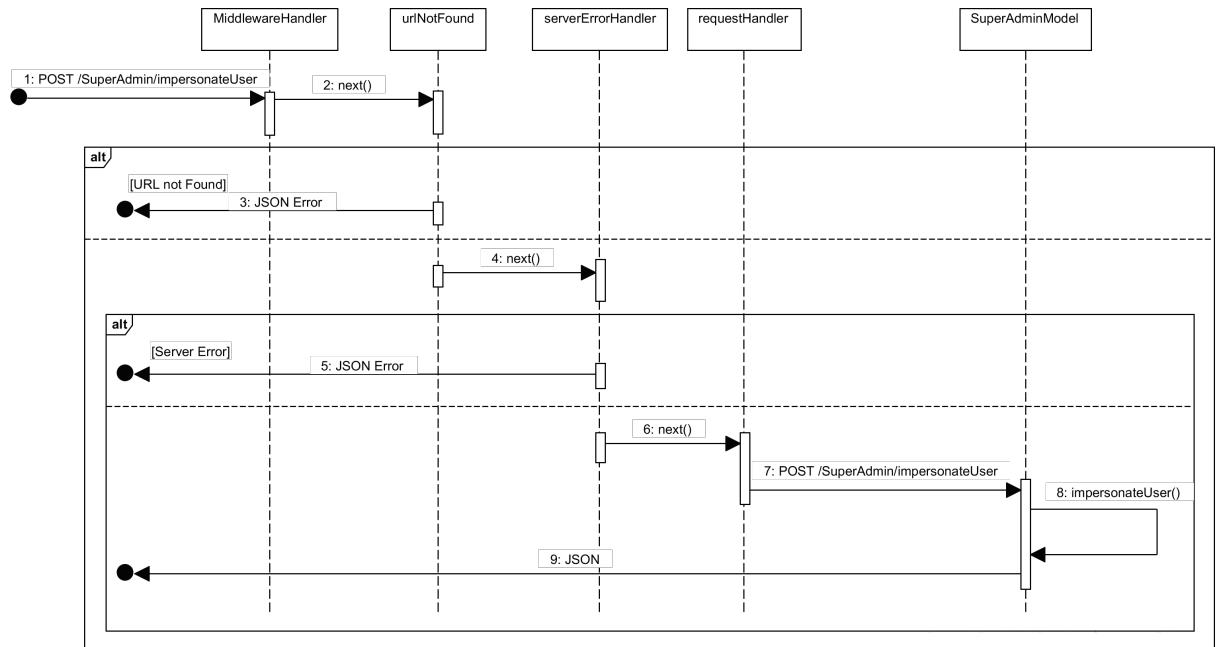


Figura 44: Diagramma di sequenza: POST /SuperAdmin/impersonateUser

### 5.2.3.3 Richieste Companies

#### Richiesta GET /Companies

- Descrizione:** Mostra i dati di tutte le aziende.
- Richiesta HTTP:** /Companies
- Metodo:** GET
- Permessi:** Super Amministratore.
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Companies. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al CompanyModel che chiama il metodo getCompanies() che richiede i dati relativi di tutte le aziende. Il CompanyModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

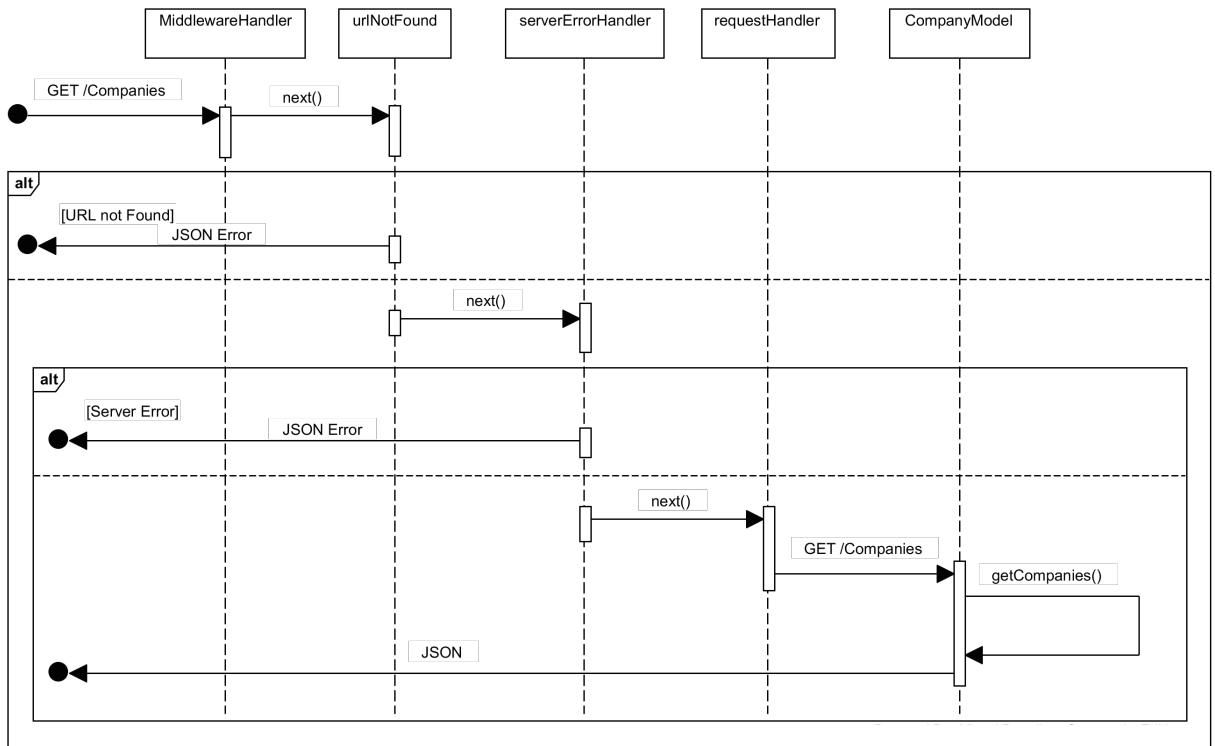


Figura 45: Diagramma di sequenza: GET /Companies

### Richiesta POST /Companies

- Descrizione:** Crea una azienda.
- Richiesta HTTP:** /Companies
- Metodo:** POST
- Permessi:** Super Amministratore.
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa /Companies. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al CompanyModel che chiama il metodo createCompany() che permette la registrazione di una azienda. Il CompanyModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

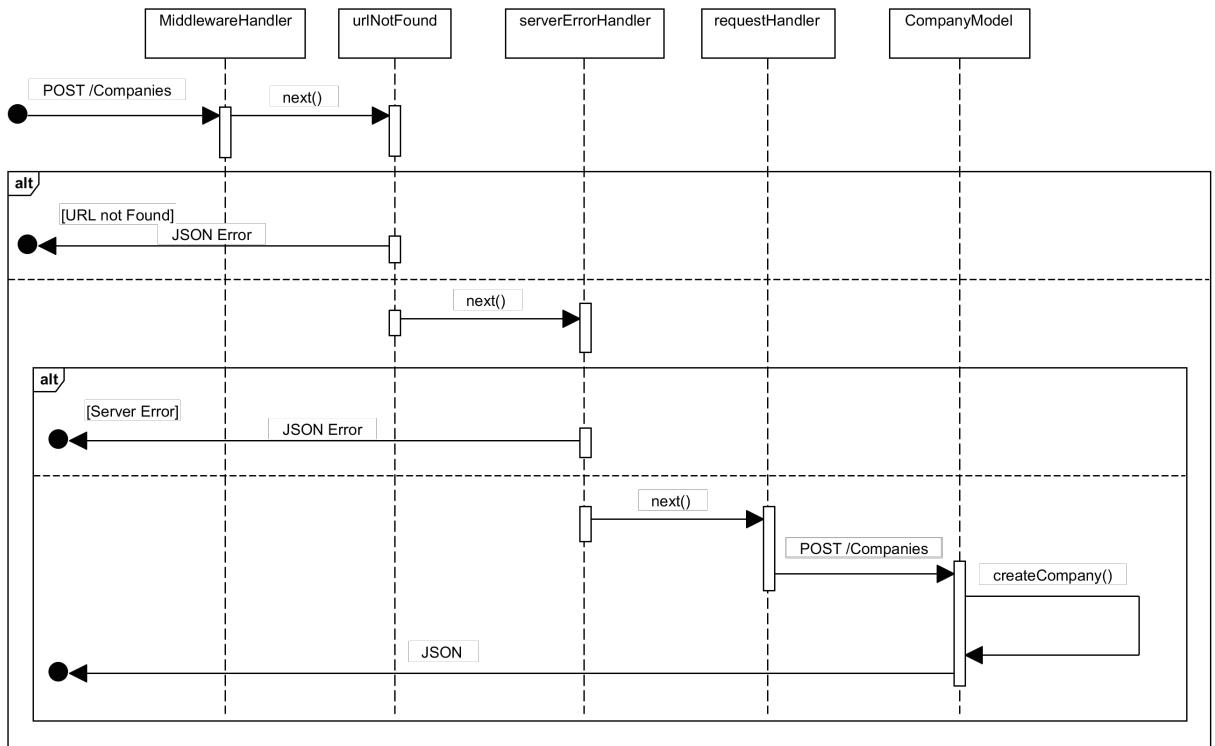


Figura 46: Diagramma di sequenza: POST /Companies

#### Richiesta GET /Companies/{companyName}

- Descrizione:** Mostra i dati di una singola azienda identificandola col parametro companyName.
- Richiesta HTTP:** /Companies/{companyName}
- Metodo:** GET
- Permessi:** Super Amministratore.
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Companies/{companyName}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al CompanyModel che chiama il metodo getCompany(name) richiede i dati di una specifica azienda. Il CompanyModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

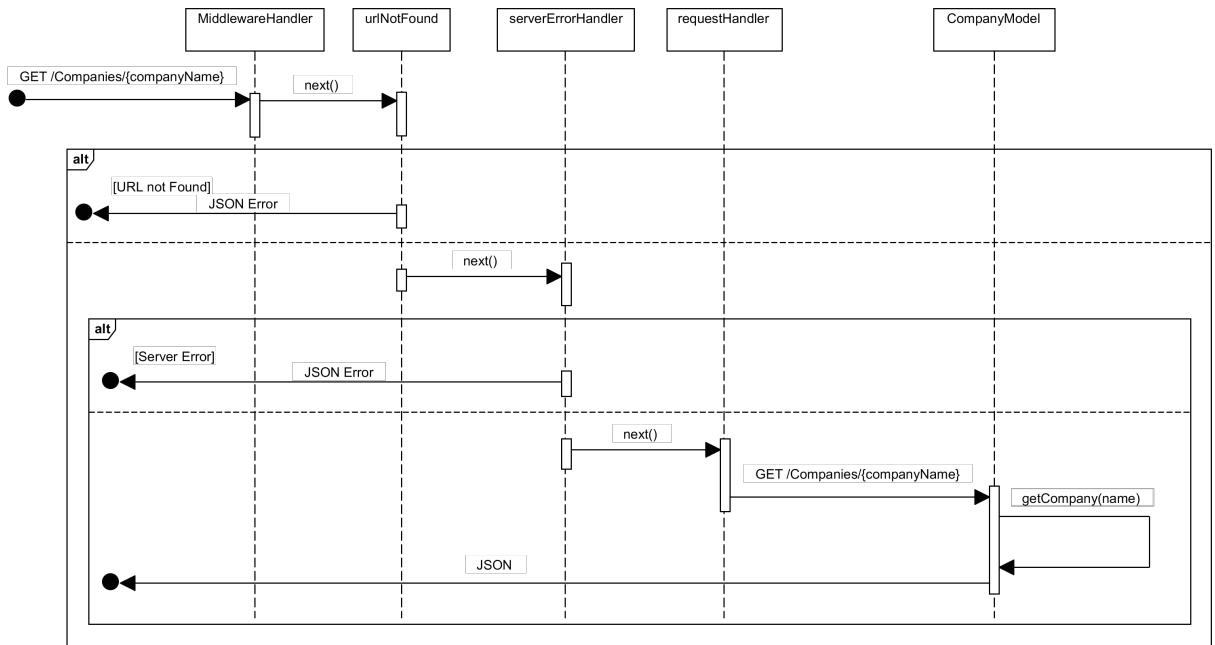


Figura 47: Diagramma di sequenza: GET /Companies/{companyName}

### Richiesta PUT /Companies

- **Descrizione:** Modifica i dati di una azienda.
- **Richiesta HTTP:** /Companies
- **Metodo:** PUT
- **Permessi:** Super Amministratore.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta PUT per la risorsa /Companies. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta al `CompanyModel` che chiama il metodo `editCompany()` che modifica i dati di una azienda. Il `CompanyModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

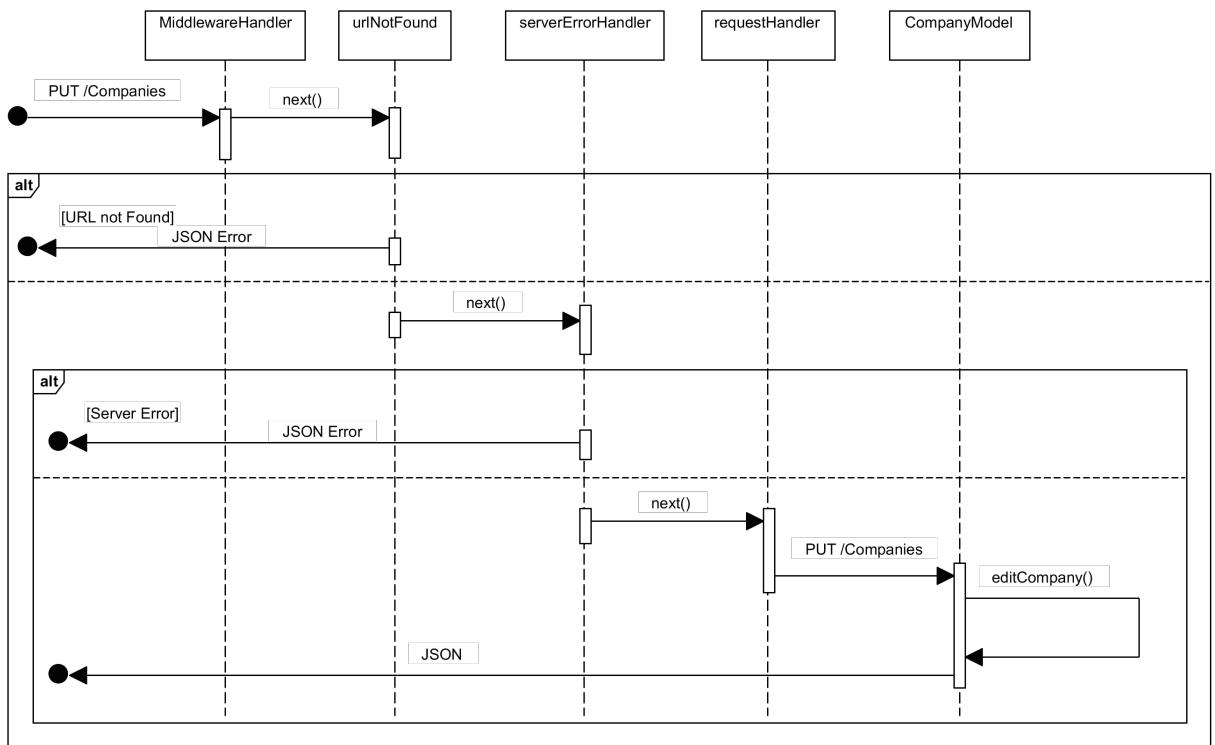


Figura 48: Diagramma di sequenza: PUT /Companies

#### Richiesta DELETE /Companies/deleteCompanies

- **Descrizione:** Elimina un'azienda.
- **Richiesta HTTP:** /Companies/deleteCompanies
- **Metodo:** DELETE
- **Permessi:** Super Amministratore.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta DELETE per la risorsa /Companies/deleteCompanies. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al CompanyModel che chiama il metodo deleteCompany() che elimina una azienda registrata sul sistema. Il CompanyModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

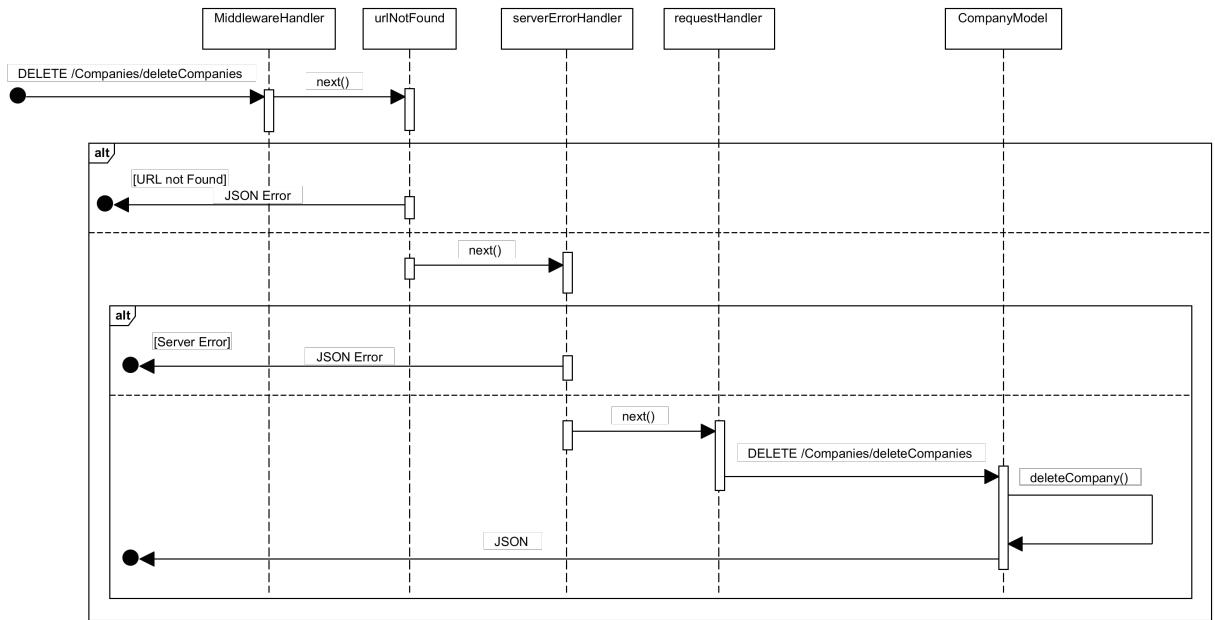


Figura 49: Diagramma di sequenza: DELETE /Companies/deleteCompanies

#### Richiesta GET /Companies/searchCompany/{value}

- **Descrizione:** Ricerca di una azienda.
- **Richiesta HTTP:** /Companies/searchCompany/{value}
- **Metodo:** GET
- **Permessi:** Super Amministratore.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Companies/searchCompany/{value}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al CompanyModel che chiama il metodo searchCompany(value) che dato un valore, ristituisce la lista di aziende che soddisfano quella specifica caratteristica. Il CompanyModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

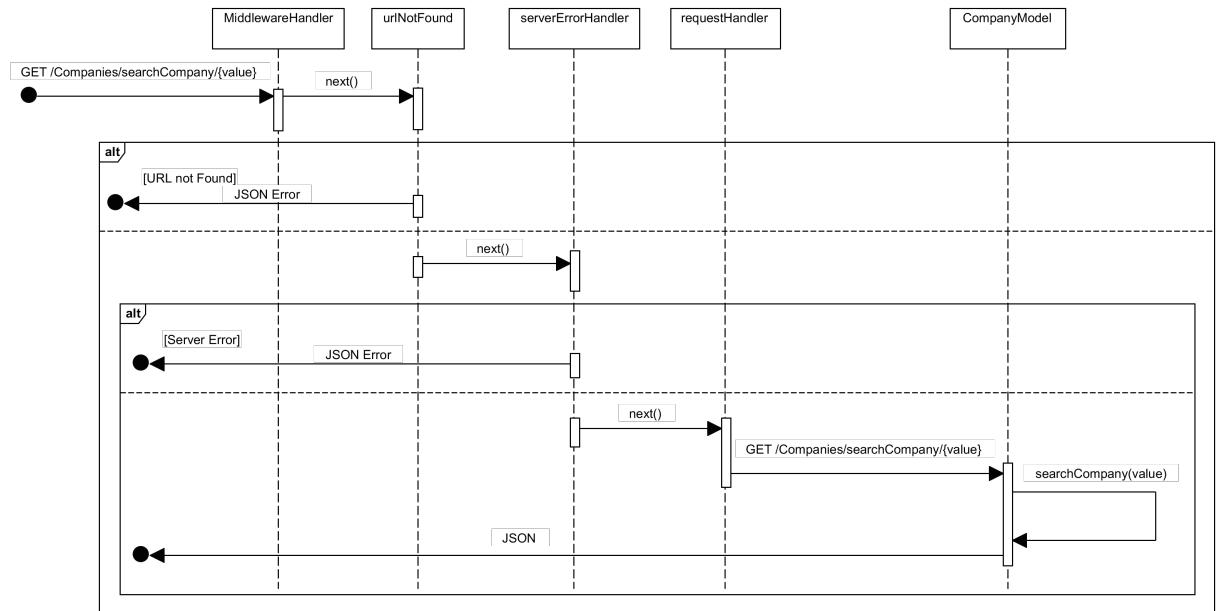


Figura 50: Diagramma di sequenza: GET /Companies/searchCompany/{value}

#### 5.2.3.4 Richieste Databases

##### Richiesta POST /ExternalDatabases

- Descrizione:** Crea una risorsa rappresentante un database esterno mongoDB.
- Richiesta HTTP:**/ExternalDatabases
- Metodo:** POST
- Permessi:** Proprietario, Amministratore.
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa /ExternalDatabases. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta all'ExternalDatabaseModel che chiama il metodo addSource() che permette di connettere un database MongoDB alla piattaforma. L'ExternalDatabaseModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

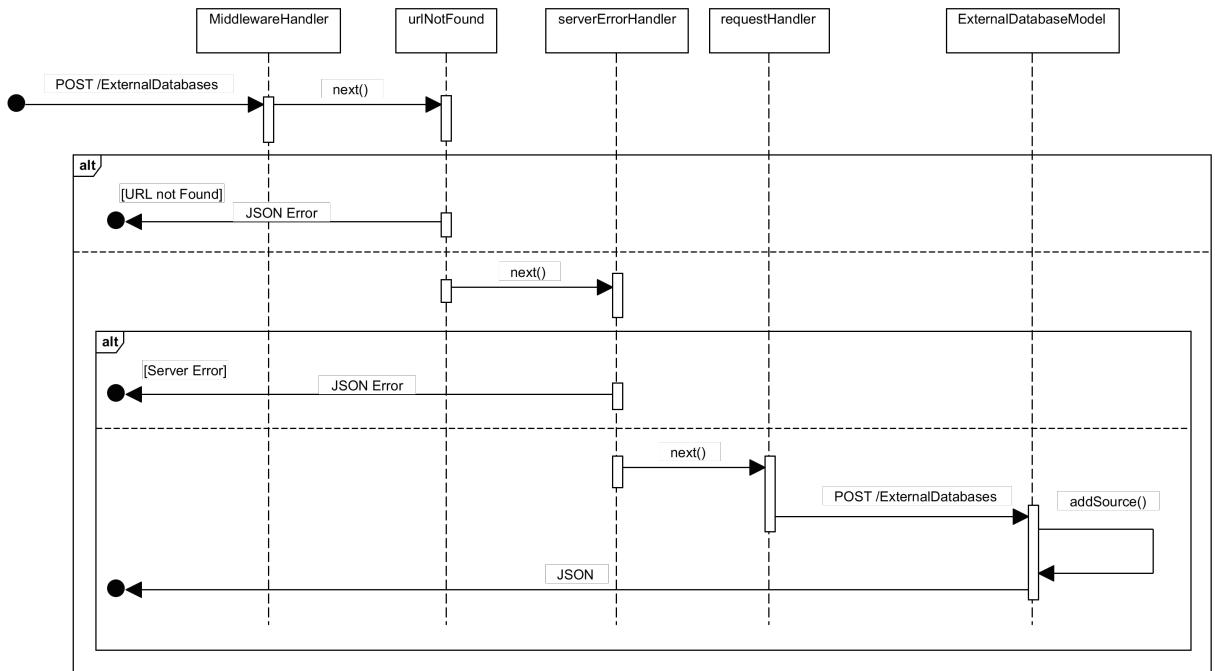
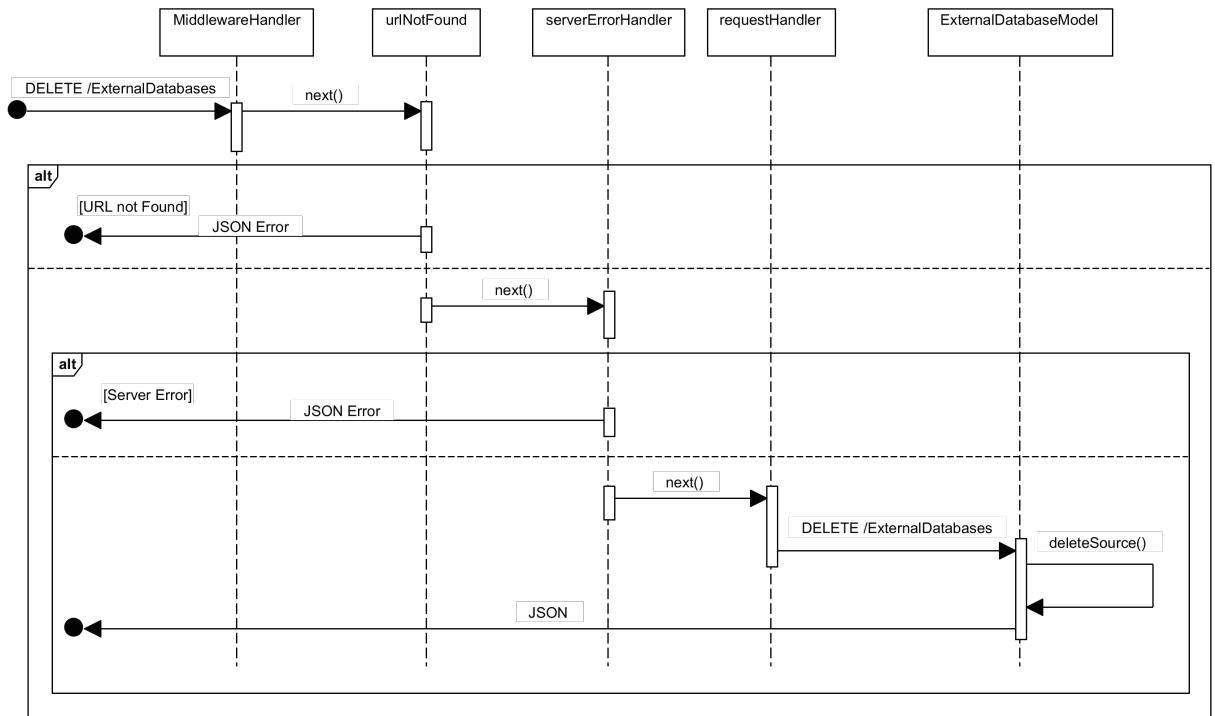


Figura 51: Diagramma di sequenza: POST /ExternalDatabases

#### Richiesta DELETE /ExternalDatabases

- Descrizione:** Elimina una risorsa rappresentante un database esterno mongoDB.
- Richiesta HTTP:** /ExternalDatabases
- Metodo:** DELETE
- Permessi:** Proprietario, Amministratore.
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta DELETE per la risorsa /ExternalDatabases. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta all'ExternalDatabaseModel che chiama il metodo deleteSource() che permette la disconnessione di un database MongoDB. L'ExternalDatabaseModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.


 Figura 52: Diagramma di sequenza: `DELETE /ExternalDatabases`

#### Richiesta GET /ExternalDatabases/{query}

- **Descrizione:** Mostra i dati dei database esterni appartenenti ad una certa azienda.
- **Richiesta HTTP:** `/ExternalDatabases/{query}`
- **Metodo:** GET
- **Permessi:** Proprietario, Amministratore.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa `/ExternalDatabases/{query}`. La richiesta viene passata al `MiddlewareHandler` il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta all'`ExternalDatabaseModel` che chiama il metodo `getData()` che restituisce i risultati di una query. L'`ExternalDatabaseModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

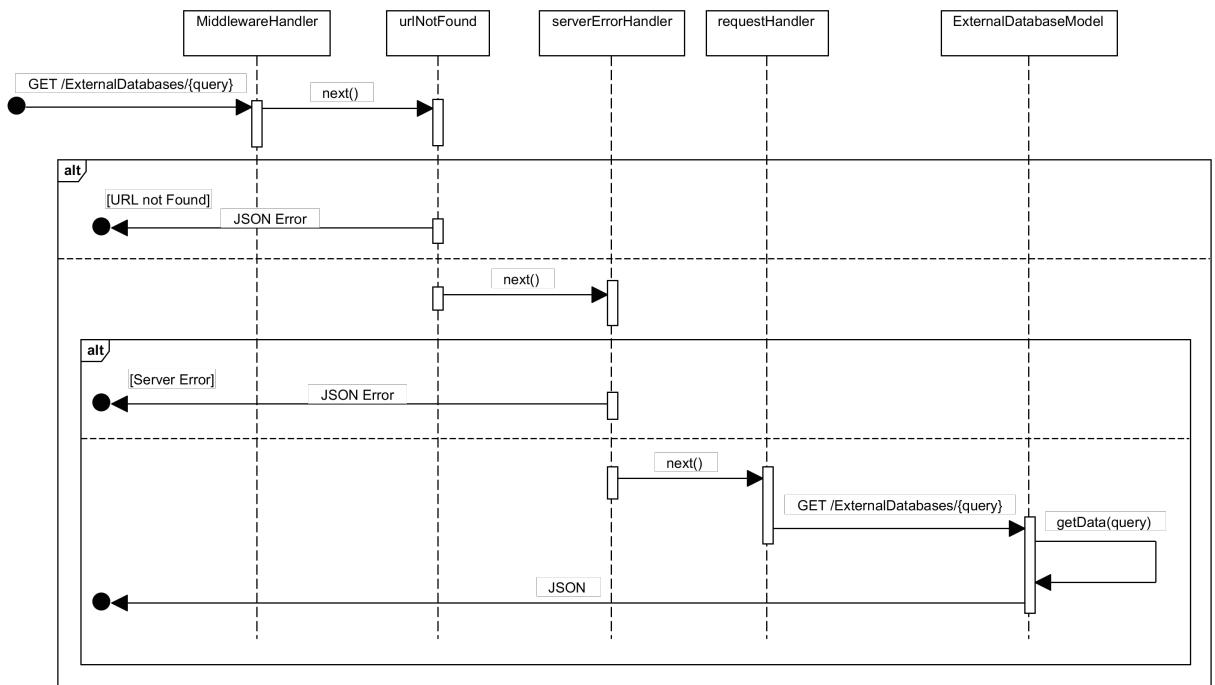


Figura 53: Diagramma di sequenza: GET /ExternalDatabases/{query}

#### Richiesta GET /ExternalDatabases/searchExternalDatabase/{value}

- **Descrizione:** Cerca un database esterno.
- **Richiesta HTTP:** /ExternalDatabases/searchExternalDatabase/value, companyName
- **Metodo:** GET
- **Permessi:** Proprietario, Amministratore.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /ExternalDatabases/searchExternalDatabase/{value}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta all'ExternalDatabaseModel che chiama il metodo searchDB(value) che dato un valore restituisce la lista di database con quella caratteristica. L'ExternalDatabaseModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

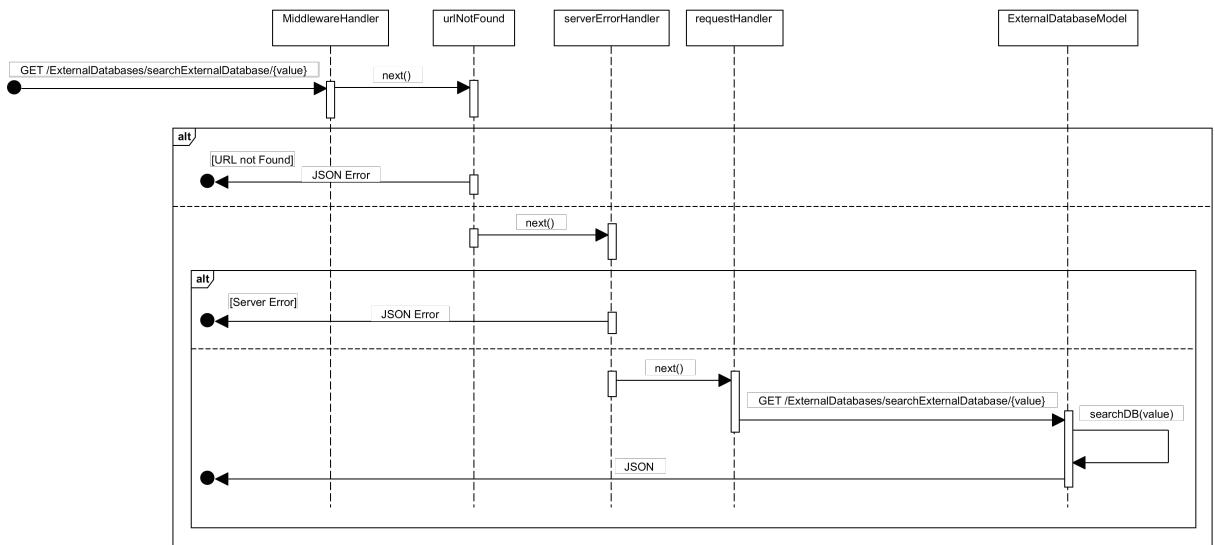


Figura 54: Diagramma di sequenza: GET /ExternalDatabases/searchExternalDatabase/{value}

#### Richiesta POST /ExternalDatabases/allowExternalDatabasesAccess

- **Descrizione:** Imposta il permesso di accesso ai dati di un database esterno.
- **Richiesta HTTP:** /ExternalDatabases/allowExternalDatabasesAccess
- **Metodo:** POST
- **Permessi:** Proprietario, Amministratore.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa /ExternalDatabases/allowExternalDatabasesAccess. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta all'ExternalDatabaseModel che chiama il metodo allowAccess() che permette ad un utente di accedere ai dati relativi ad un database. L'ExternalDatabaseModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

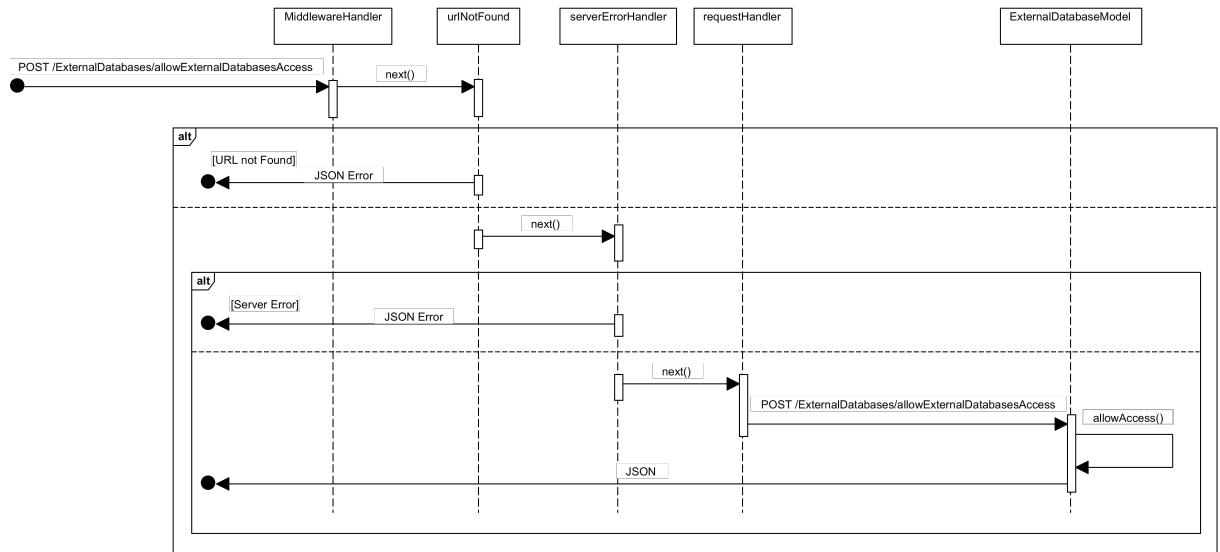


Figura 55: Diagramma di sequenza: POST /ExternalDatabases/allowExternalDatabasesAccess

#### Richiesta POST /ExternalDatabases/denyExternalDatabaseAccess

- **Descrizione:** Imposta negativamente i permessi di accesso ad un database esterno, da parte di un utente.
- **Richiesta HTTP:** /ExternalDatabases/{id}
- **Metodo:** POST
- **Permessi:** Proprietario, Amministratore.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa /ExternalDatabases/denyExternalDatabaseAccess. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta all'ExternalDatabaseModel che chiama il metodo denyAccess() che vieta ad un utente l'accesso ai dati relativi ad un database. L'ExternalDatabaseModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

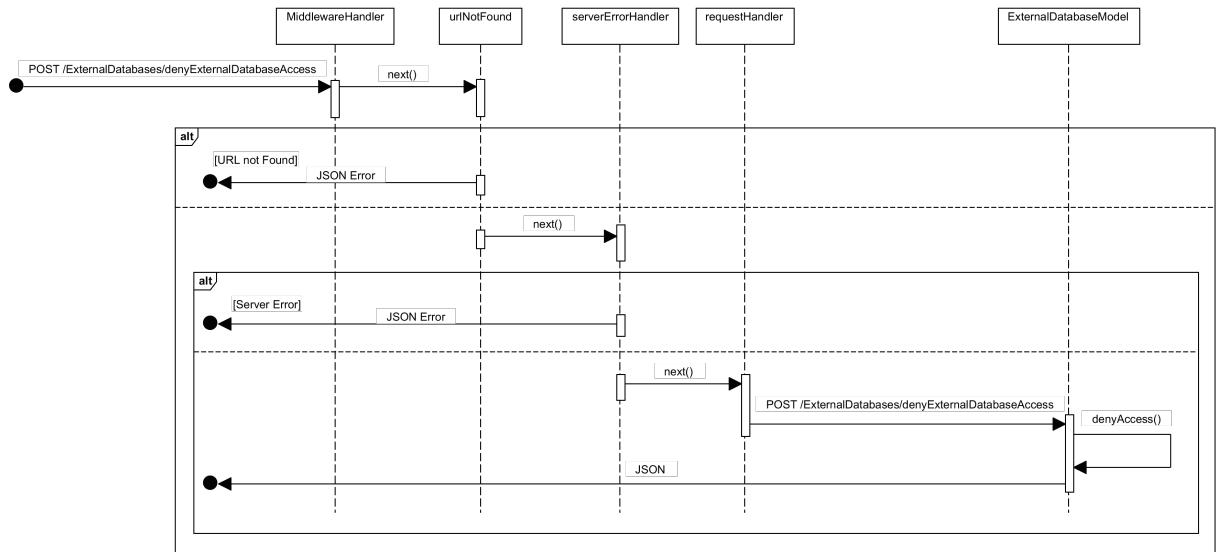


Figura 56: Diagramma di sequenza: POST /ExternalDatabases/denyExternalDatabaseAccess

### 5.2.3.5 Richieste Collections

#### Richiesta GET /Collections

- **Descrizione:** Mostra la lista delle Collection.
- **Richiesta HTTP:** /Collections
- **Metodo:** GET
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Collections. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al CollectionModel che chiama il metodo getCollections() che chiede di dati di tutte le Collections di un utente. Il CollectionModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

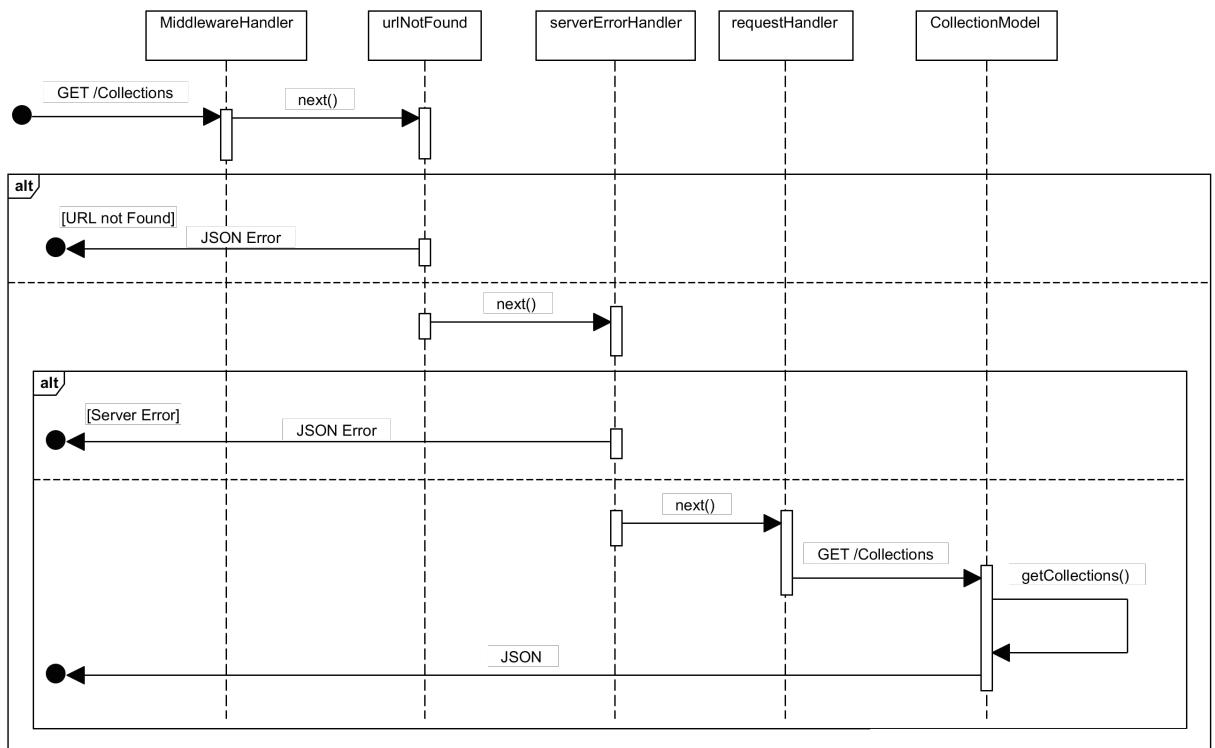


Figura 57: Diagramma di sequenza: GET /Collections

### Richiesta POST /Collections

- **Descrizione:** Creazione di una Collection
- **Richiesta HTTP:** /Collections
- **Metodo:** POST
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa /Collections. La richiesta viene passata al `MiddlewareHandler` il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta al `CollectionModel` che chiama il metodo `createCollection()` che permette la creazione di una Collection. Il `CollectionModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

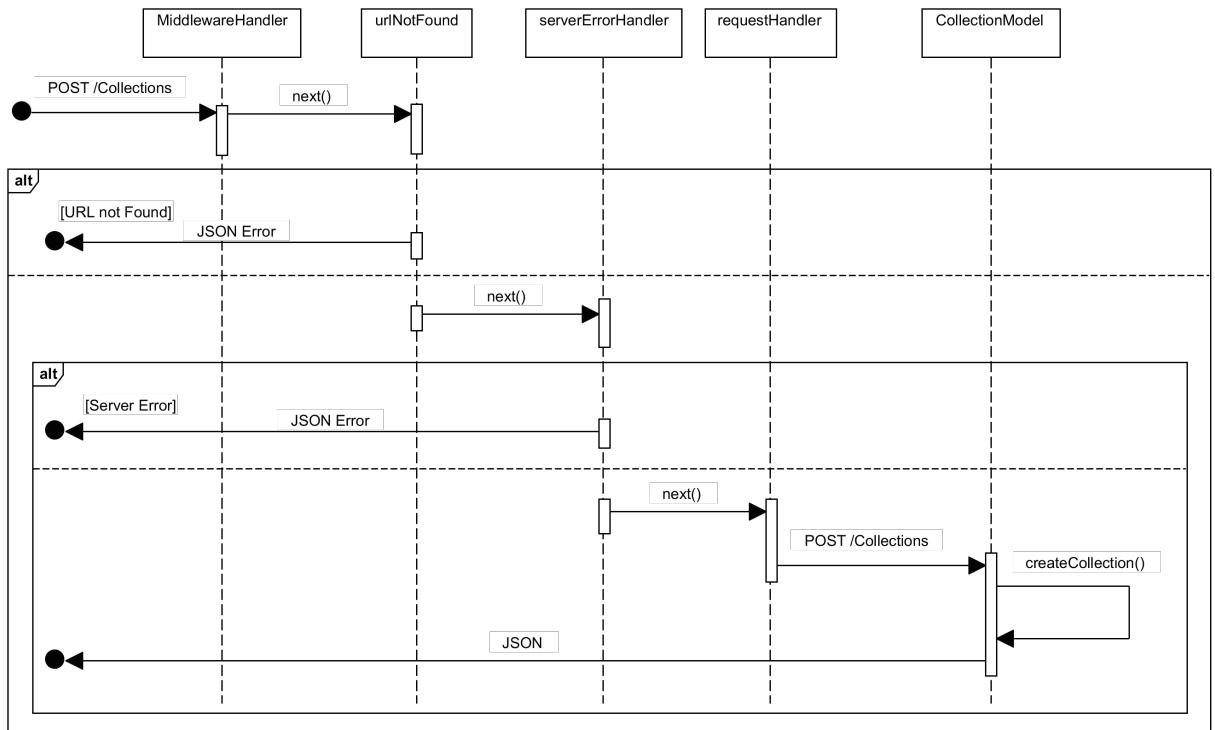


Figura 58: Diagramma di sequenza: POST /Collections

### Richiesta PUT /Collections

- **Descrizione:** Modifica di una Collection
- **Richiesta HTTP:** /Collections
- **Metodo:** PUT
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta PUT per la risorsa /Collections. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al CollectionModel che chiama il metodo editCollection() che permette di modificare una Collection. Il CollectionModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

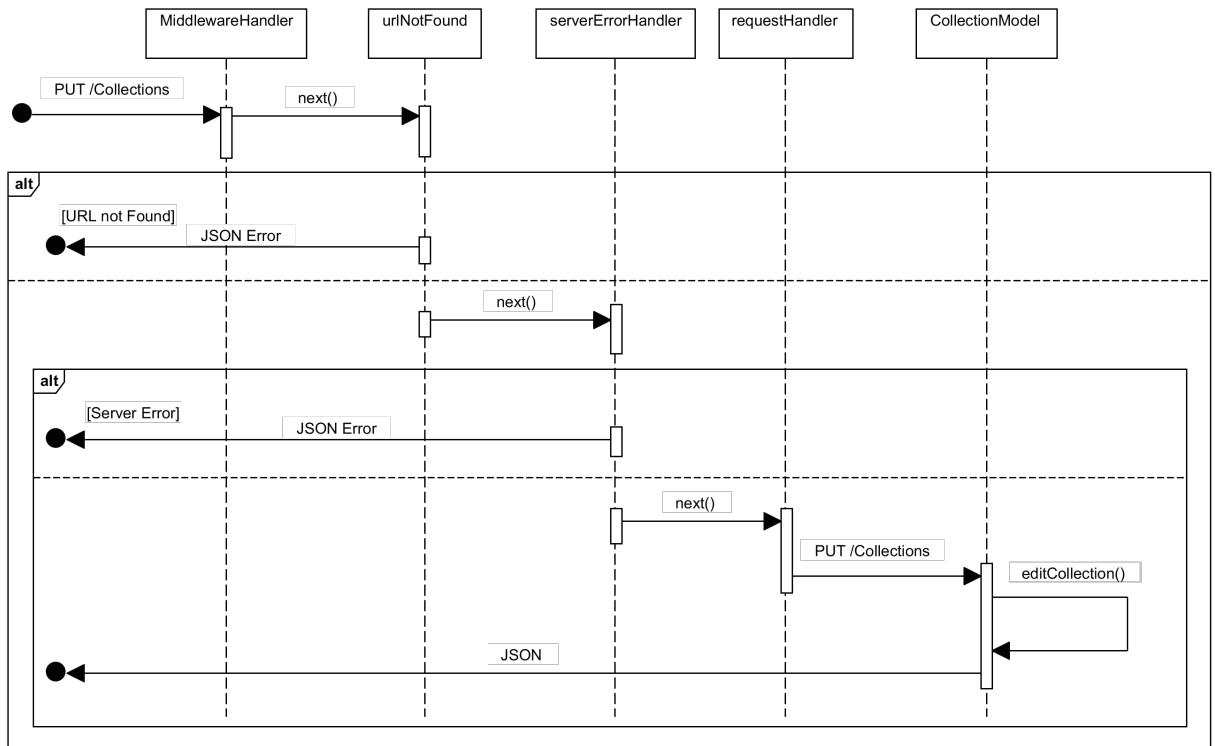
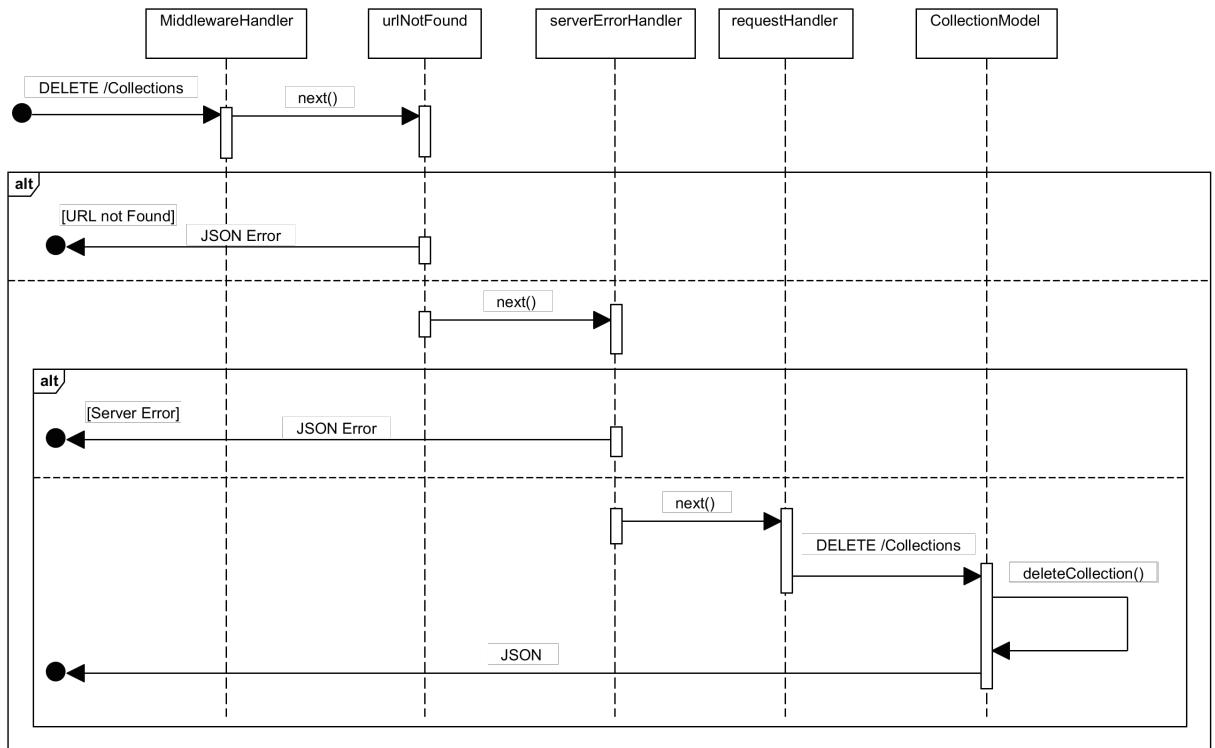


Figura 59: Diagramma di sequenza: PUT /Collections

### Richiesta DELETE /Collections

- **Descrizione:** Eliminazione di una Collection mediante il suo identificativo.
- **Richiesta HTTP:** /Collections
- **Metodo:** DELETE
- **Permessi:** Utente autenticato.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta DELETE per la risorsa /Collections. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al CollectionModel che chiama il metodo deleteCollection() che elimina una Collection. Il CollectionModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.


 Figura 60: Diagramma di sequenza: `DELETE /Collections`

#### Richiesta GET /Collections/retrieveCollectionDSLIS/{id}

- **Descrizione:** Ritorna il codice della Collection
- **Richiesta HTTP:** `/Collections/retrieveCollectionDSLIS/{id}`
- **Metodo:** GET
- **Permessi:** Utente autenticato.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa `/Collections/retrieveCollectionDSLIS/{id}`. La richiesta viene passata al `MiddlewareHandler` il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta al `CollectionModel` che chiama il metodo `retrieveCollection(id)` che ritorna il codice di una Collection. Il `CollectionModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

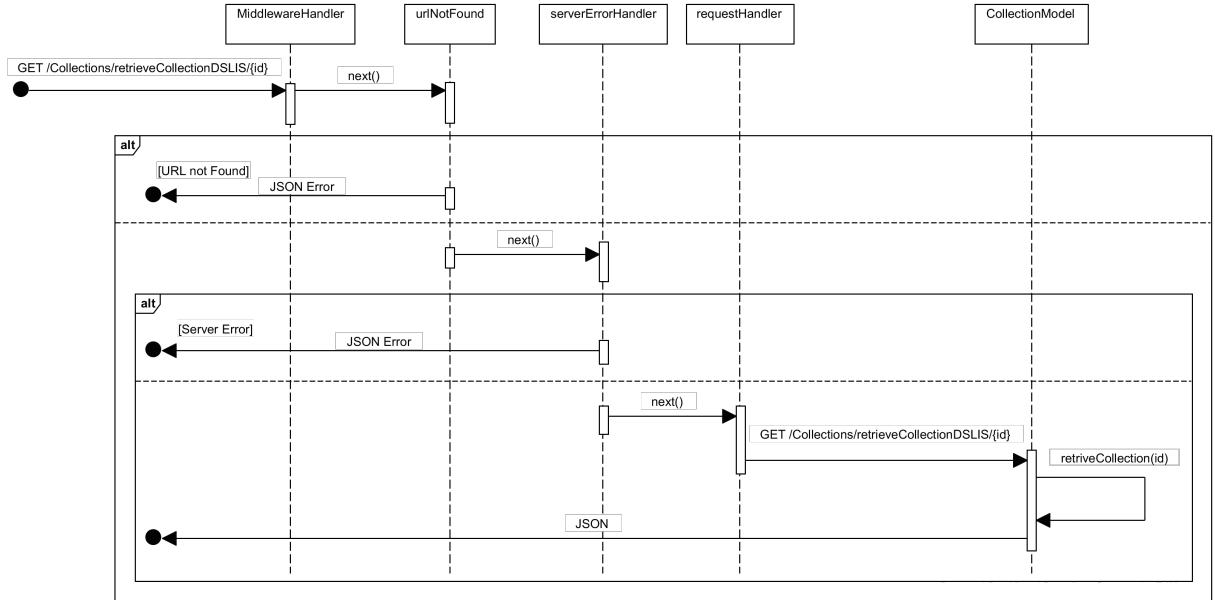


Figura 61: Diagramma di sequenza: GET /Collections/retrieveCollectionDSLIS/{id}

#### Richiesta GET /Collections/searchCollections/{value, email}

- **Descrizione:** Ricerca di una Collection tramite il suo identificativo.
- **Richiesta HTTP:** /Collections/searchCollections/{value, email}
- **Metodo:** GET
- **Permessi:** Utente autenticato.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Collections/searchCollections/{value, email}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta al `CollectionModel` che chiama il metodo `searchCollection(value, email)` che effettua la ricerca di una specifica Collection dato un valore, mostrando solamente quelle a cui l'utente con l'email corrispondente ha accesso. Il `CollectionModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

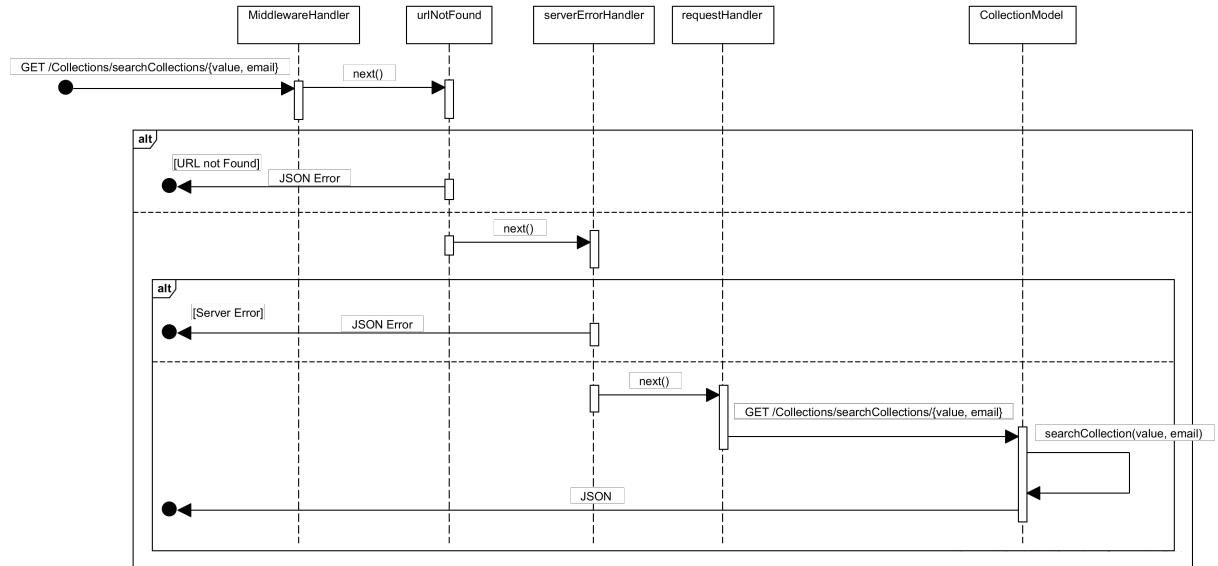


Figura 62: Diagramma di sequenza: GET /Collections/searchCollections/{value, email}

#### Richiesta GET /Collections/ExecuteCollection/{id}

- **Descrizione:** Ritorna la lista dei Document appartenenti ad una specifica Collection, oltre ad altri campi.
- **Richiesta HTTP:** /Collections/ExecuteCollection/{id}
- **Metodo:** GET
- **Permessi:** Utente autenticato.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Collections/ExecuteCollection/{id}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al CollectionModel che chiama il metodo executeCollection(id) che dato l'identificativo esegue una specifica Collection. Il CollectionModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

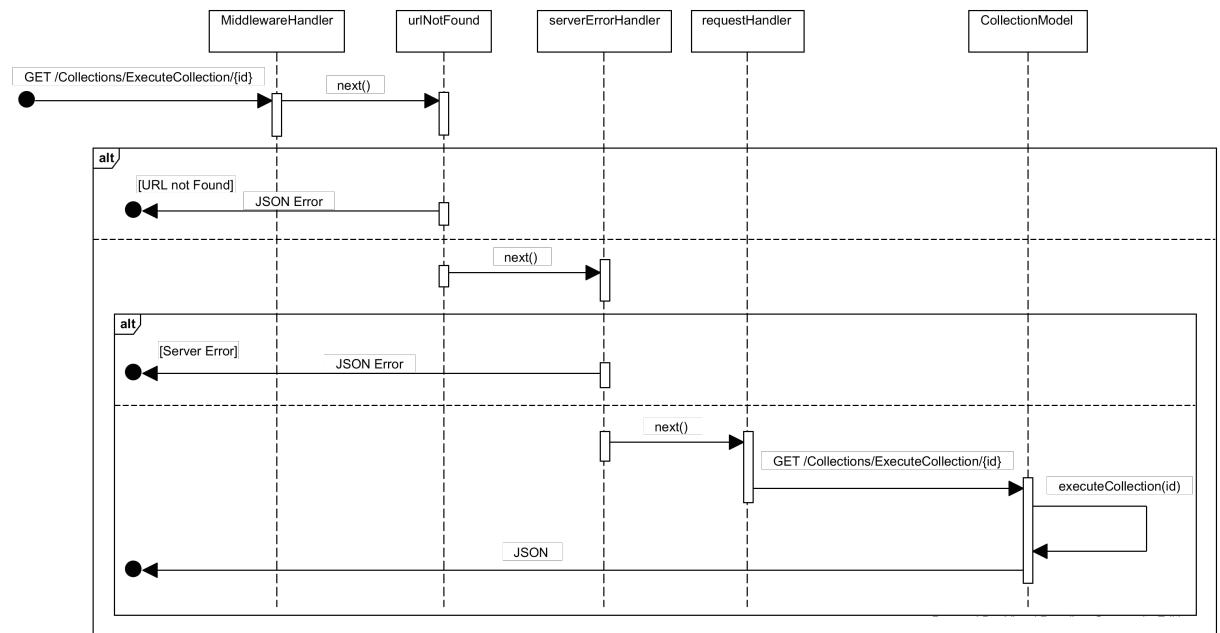


Figura 63: Diagramma di sequenza: GET /Collections/ExecuteCollection/{id}

#### Richiesta GET /Collections/SendEmail/{id, format}

- Descrizione:** Permette di inviare tramite email la struttura dei dati della Collection selezionata in un formato a scelta tra JSON e CSV.
- Richiesta HTTP:** /Collections/SendEmail/{id, format}
- Metodo:** GET
- Permessi:** Utente autenticato
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Collections/SendEmail{id, format}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al CollectionModel che chiama il metodo sendEmail(id, format) che invia una mail contenente la struttura dati di una specifica Collection in un formato a scelta (JSON o CSV). Il CollectionModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

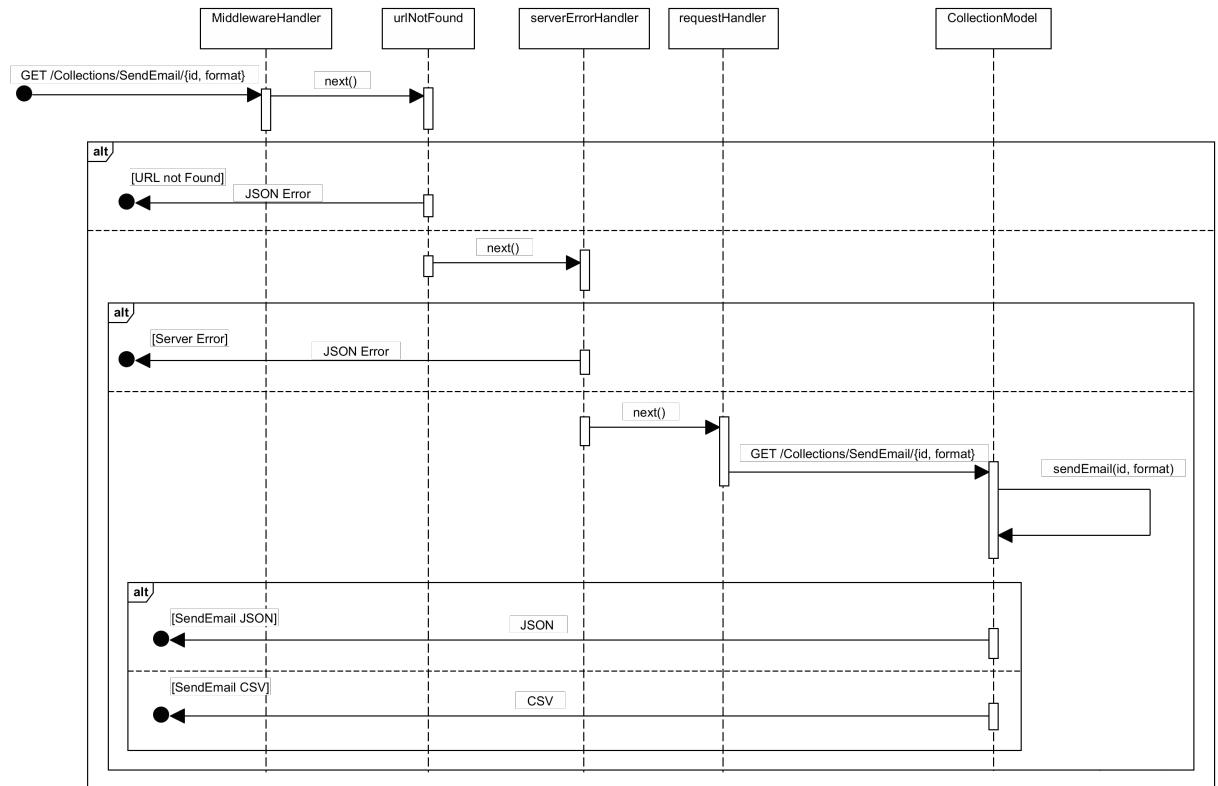


Figura 64: Diagramma di sequenza: GET /Collections/SendEmail/{id, format}

#### Richiesta GET /Collections/Export/{id, format}

- Descrizione:** Permette di esportare la struttura dei dati della Collection selezionata in un formato a scelta tra JSON e CSV.
- Richiesta HTTP:** /Collections/Export/{id, format}
- Metodo:**
- Permessi:** Utente autenticato.
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Collections/Export/{id, format}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al CollectionModel che chiama il metodo export(id) che esporta la struttura dati di una specifica Collection in un formato a scelta (JSON o CSV). Il CollectionModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

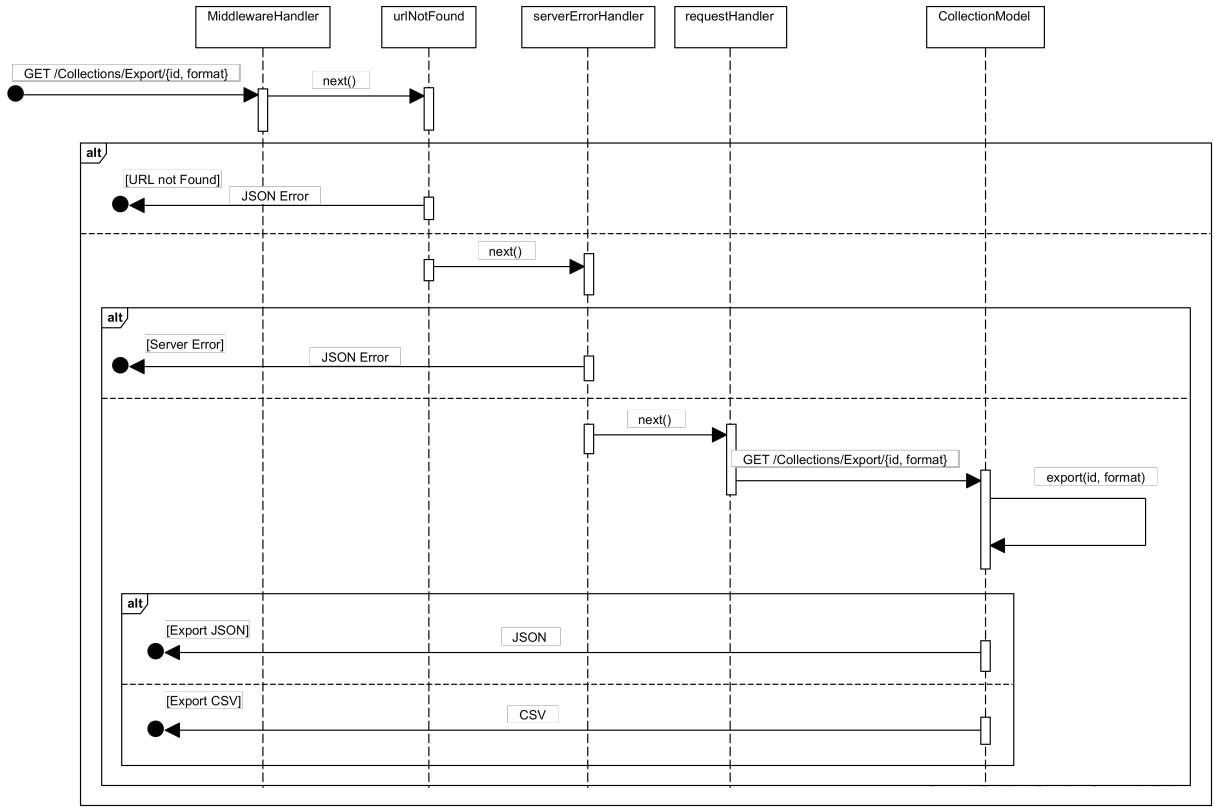


Figura 65: Diagramma di sequenza: GET /Collections/Export/{id, format}

#### Richiesta GET /Collections/exportCollectionDSLIS/{id}

- Descrizione:** Permette di esportare un DSLIS rappresentante una Collection.
- Richiesta HTTP:** /Collections/exportCollectionDSLIS/{id}
- Metodo:** GET
- Permessi:** Proprietario, Amministratore, Membro
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Collections/exportCollectionDSLIS/{id}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta al `CollectionModel` che chiama il metodo `exportDSLIS(id)` che esporta il DSLIS relativo ad una specifica Collection in formato JSON. Il `CollectionModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

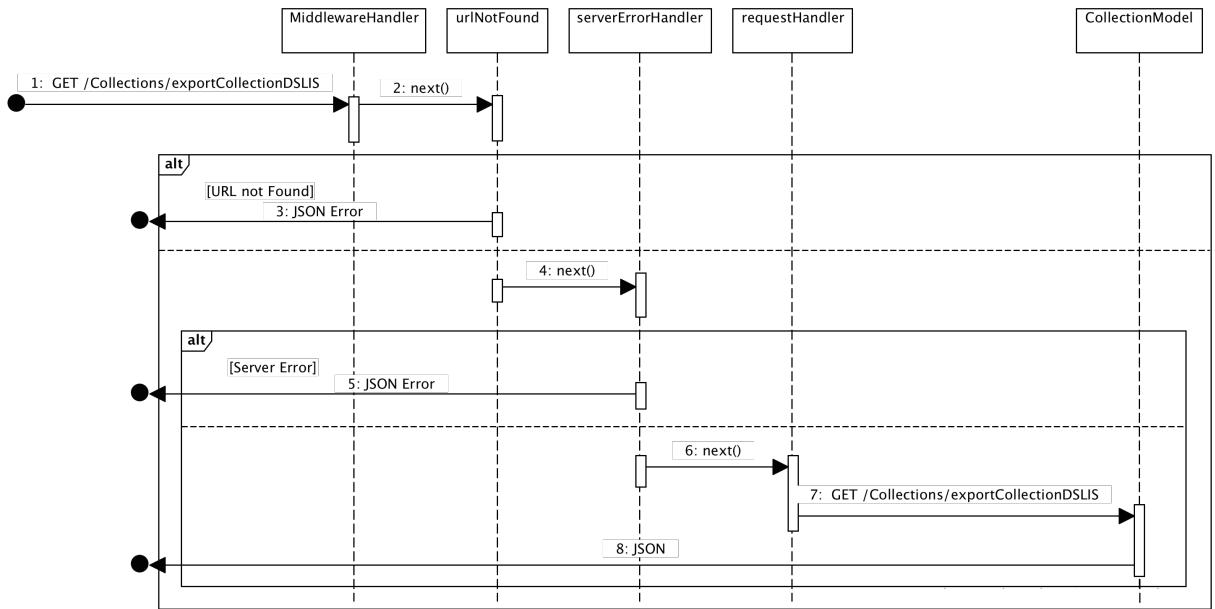


Figura 66: Diagramma di sequenza: `GET /Collections/exportCollectionDSLIS/{id}`

#### Richiesta POST /Collections/importCollectionDSLIS

- **Descrizione:** Permette di importare un DSLIS rappresentante una Collection.
- **Richiesta HTTP:** `/Collections/importCollectionDSLIS`
- **Metodo:** POST
- **Permessi:** Proprietario, Amministratore, Membro
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa `/Collections/importCollectionDSLIS`. La richiesta viene passata al `MiddlewareHandler` il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta al `CollectionModel` che chiama il metodo `importDSLIS()` che importa il DSLIS relativo ad una specifica Collection in formato JSON. Il `CollectionModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

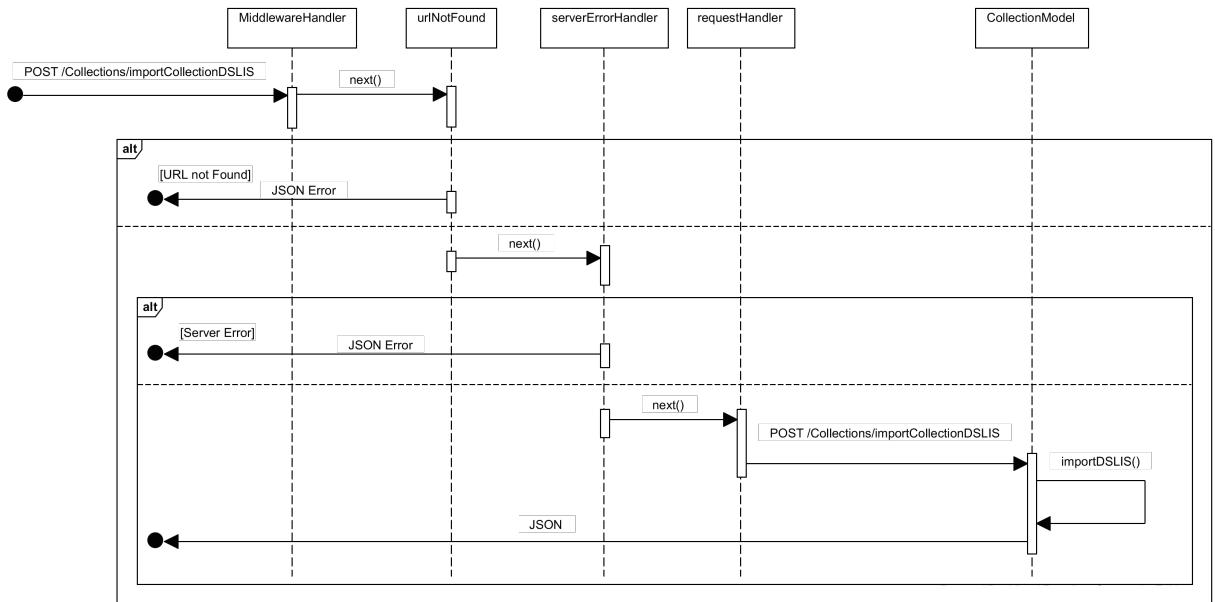


Figura 67: Diagramma di sequenza: POST /Collections/importCollectionDSLIS

#### 5.2.3.6 Richieste Documents

##### Richiesta GET /Documents

- **Descrizione:** Mostra la lista dei Documenti.
- **Richiesta HTTP:** /Documents
- **Metodo:** GET
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Documents. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al DocumentModel che chiama il metodo getDocuments() che mostra la lista dei Documents di un utente. Il DocumentModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

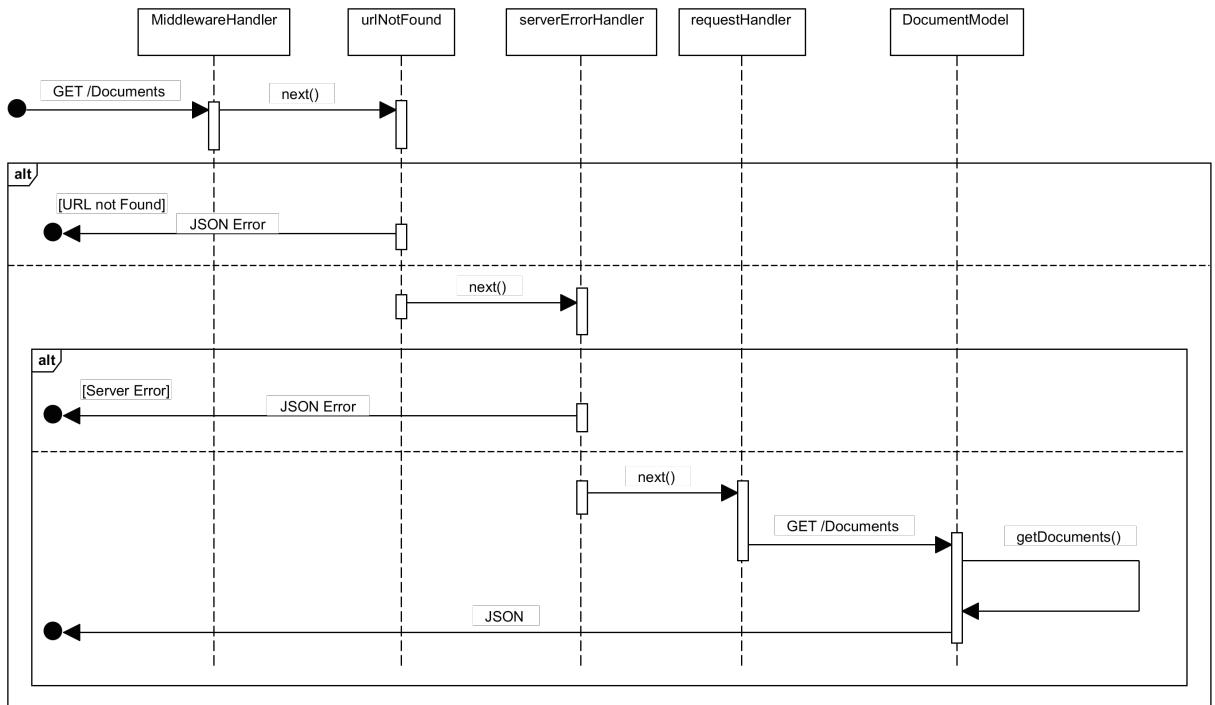


Figura 68: Diagramma di sequenza: GET /Documents

## Richiesta POST /Documents

- **Descrizione:** Creazione di un Document.
  - **Richiesta HTTP:** /Documents
  - **Metodo:** POST
  - **Permessi:** Utente autenticato
  - **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa /Documents. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al DocumentModel che chiama il metodo createDocument() che permette di creare un Document. Il DocumentModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

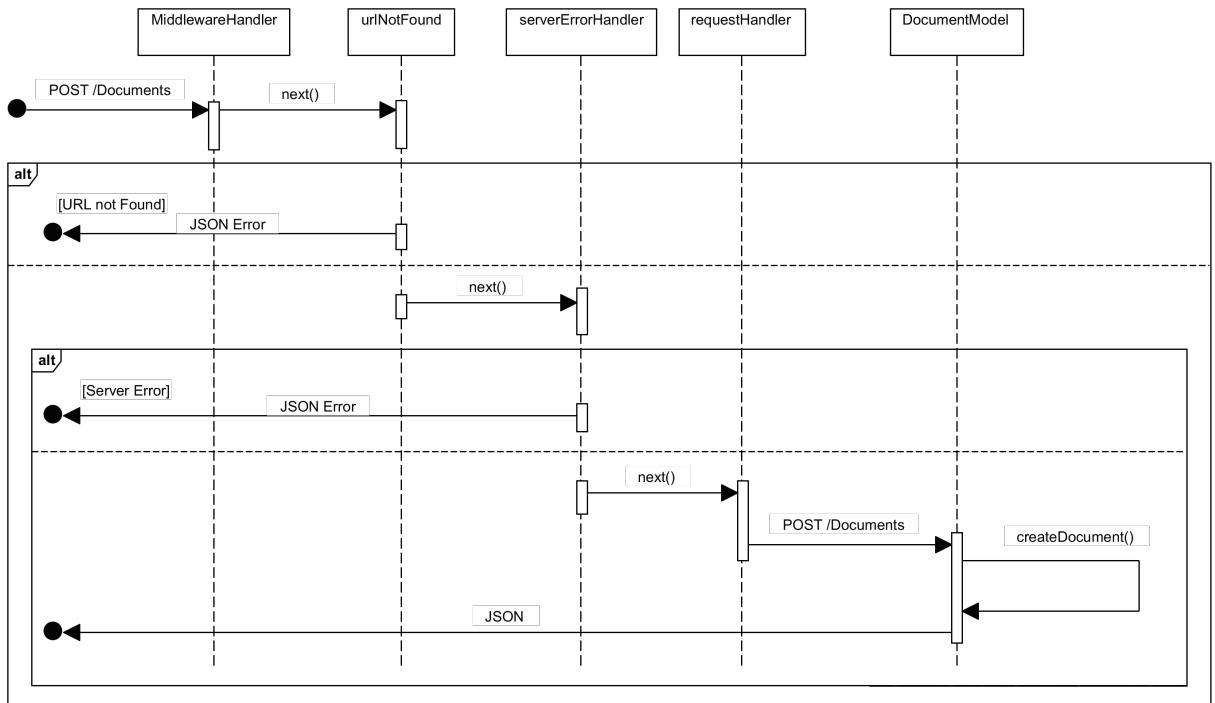
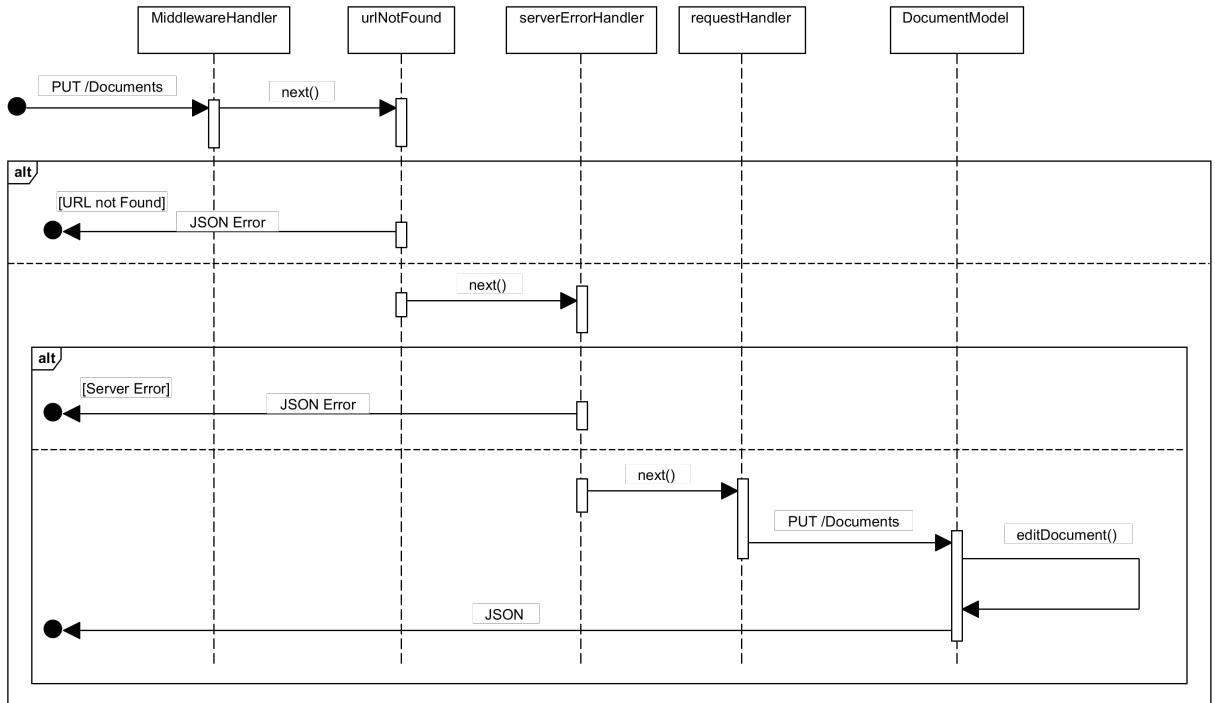


Figura 69: Diagramma di sequenza: POST /Documents

### Richiesta PUT /Documents

- **Descrizione:** Modifica di un Document
- **Richiesta HTTP:** /Documents
- **Metodo:** PUT
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta PUT per la risorsa /Documents. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al DocumentModel che chiama il metodo editDocument() che permette di modificare un Document esistente. Il DocumentModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.


 Figura 70: Diagramma di sequenza: `PUT /Documents`

#### Richiesta `DELETE /Documents`

- **Descrizione:** Eliminazione di un Document tramite identificativo.
- **Richiesta HTTP:** `/Documents`
- **Metodo:** `DELETE`
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta `DELETE` per la risorsa `/Documents`. La richiesta viene passata al `MiddlewareHandler` il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato `JSON`. Se non vi sono errori il `requestHandler` passa la richiesta al `DocumentModel` che chiama il metodo `deleteDocument()` che permette di eliminare un Document esistente. Il `DocumentModel` si occupa di ritornare un `JSON` informando il client sull'esito dell'operazione.

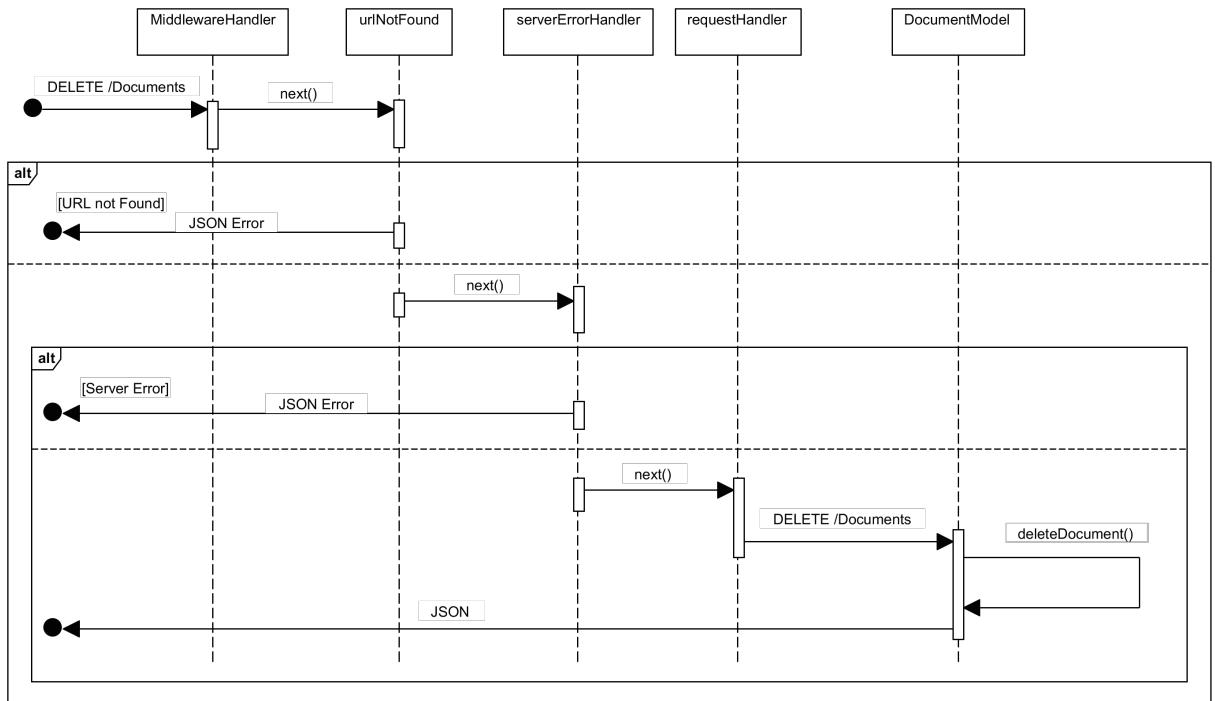


Figura 71: Diagramma di sequenza: `DELETE /Documents`

#### Richiesta GET /Documents/searchDocuments/{value, email}

- **Descrizione:** Ricerca di un Document.
- **Richiesta HTTP:** `/Documents/searchDocuments/{value, email}`
- **Metodo:** GET
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa `/Documents/searchDocuments/{value, email}`. La richiesta viene passata al `MiddlewareHandler` il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta al `DocumentModel` che chiama il metodo `searchDocument(value, email)` che dato un valore effettua la ricerca dei Document, mostrando solamente quelli a cui l'utente con l'email corrispondente ha accesso. Il `DocumentModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

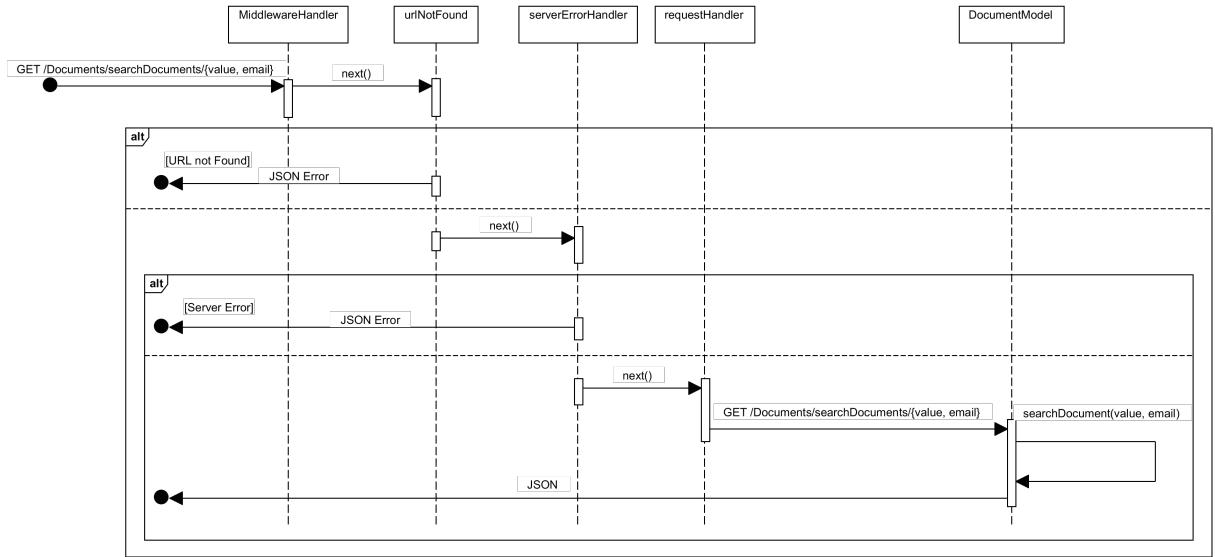


Figura 72: Diagramma di sequenza: GET /Documents/searchDocuments/{value, email}

#### Richiesta GET /Documents/ExecuteDocument/{id}

- **Descrizione:** Ritorna la lista degli attributi del Document.
- **Richiesta HTTP:** /Documents/ExecuteDocument/{id}
- **Metodo:** GET
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta PUT per la risorsa /Documents/ExecuteDocument/{id}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al DocumentModel che chiama il metodo executeDocument(id) che dato un id ritorna la lista degli attributi relati al Document cercato. Il DocumentModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

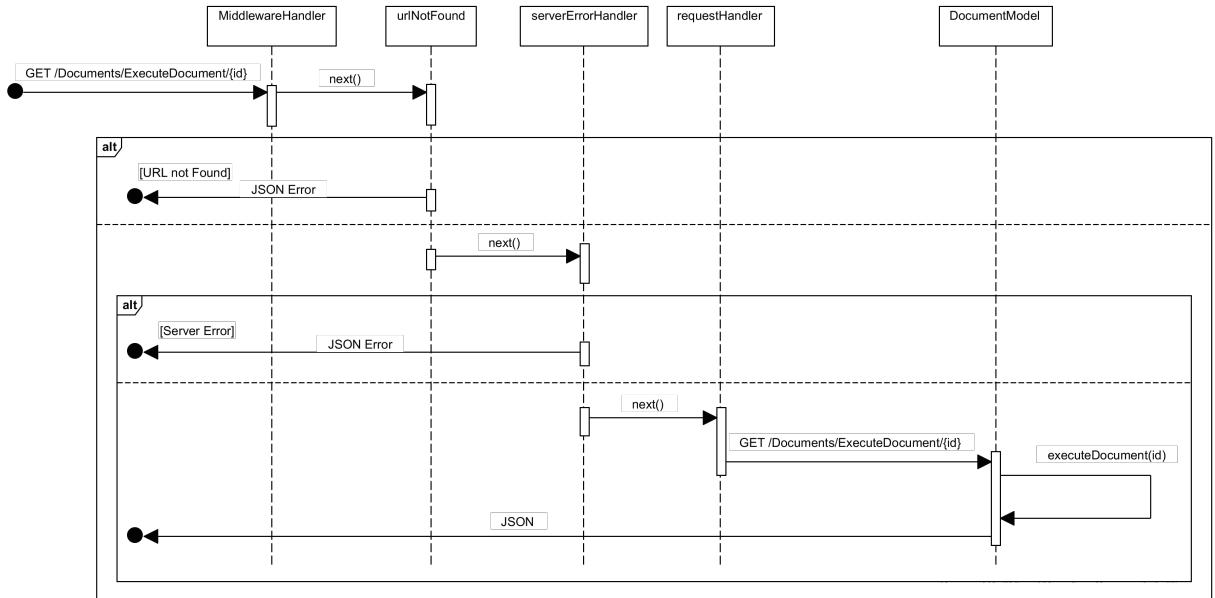


Figura 73: Diagramma di sequenza: GET /Documents/ExecuteDocument/{id}

#### Richiesta GET /Documents/ExecuteDocument/{Collection}/{id}

- Descrizione:** Ritorna la lista degli attributi del Document appartenente alla CollectionName.
- Richiesta HTTP:** /Documents/ExecuteDocument/{Collection}/{id}
- Metodo:** GET
- Permessi:** Utente autenticato
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Documents/ExecuteDocument/{Collection}/{id}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al DocumentModel che chiama il metodo executeDocument(collection,id) che dato un id e una Collection ritorna la lista degli attributi del Document appartenenti alla Collection. Il DocumentModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

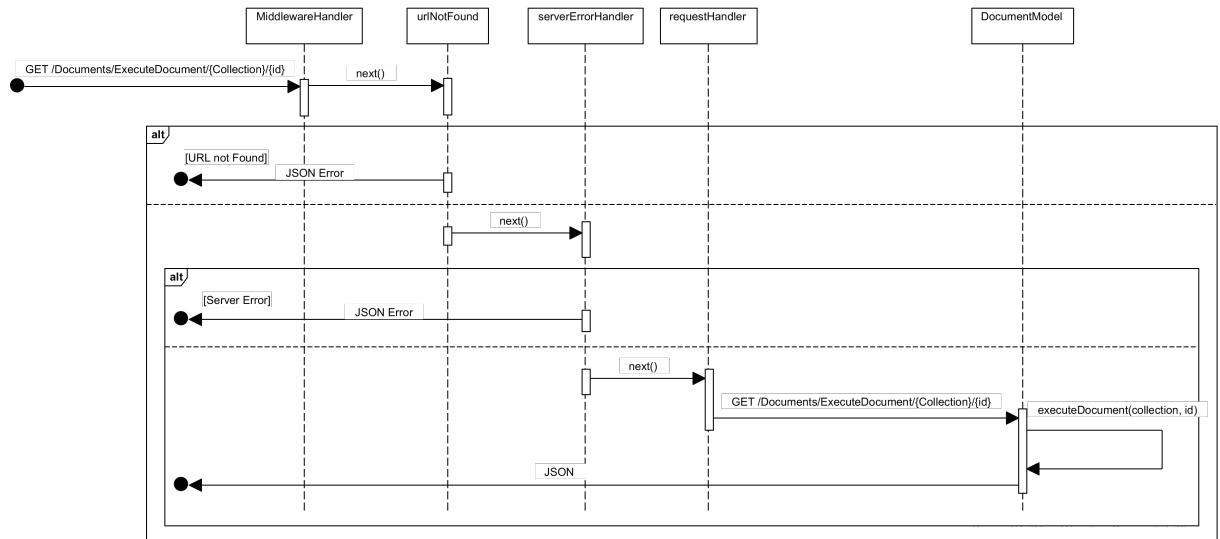


Figura 74: Diagramma di sequenza: GET /Documents/ExecuteDocument/{Collection}/{id}

#### Richiesta GET /Documents/SendEmail/{id, format}

- Descrizione:** Permette di inviare tramite email la struttura dei dati del Document selezionato in un formato a scelta tra JSON e CSV.
- Richiesta HTTP:** /Documents/SendEmail/{id, format}
- Metodo:** GET
- Permessi:** Utente autenticato.
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Documents/SendEmail/{id, format}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al DocumentModel che chiama il metodo sendEmail(id, format) che dato un id esporta il Document offrendo come scelta il formato JSON o CSV. Il DocumentModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

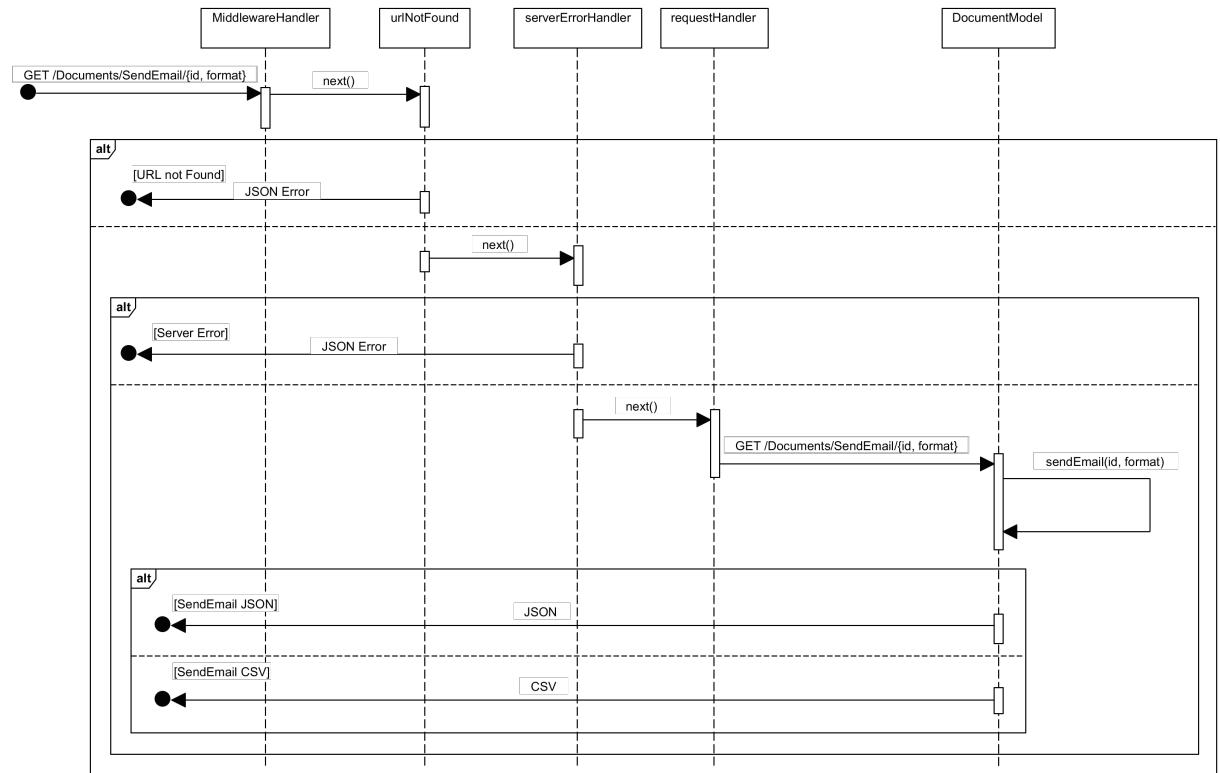


Figura 75: Diagramma di sequenza: GET /Documents/SendEmail/{id, format}

#### Richiesta GET /Documents/Export/{id, format}

- Descrizione:** Permette di esportare la struttura dei dati del Document selezionato in un formato a scelta tra JSON e CSV.
- Richiesta HTTP:** /Documents/Export/{id, format}
- Metodo:** GET
- Permessi:** Utente autenticato
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Documents/Export/{id, format}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al DocumentModel che chiama il metodo export(id, format) che dato un id esporta il Document offrendo come scelta il formato JSON o CSV. Il DocumentModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

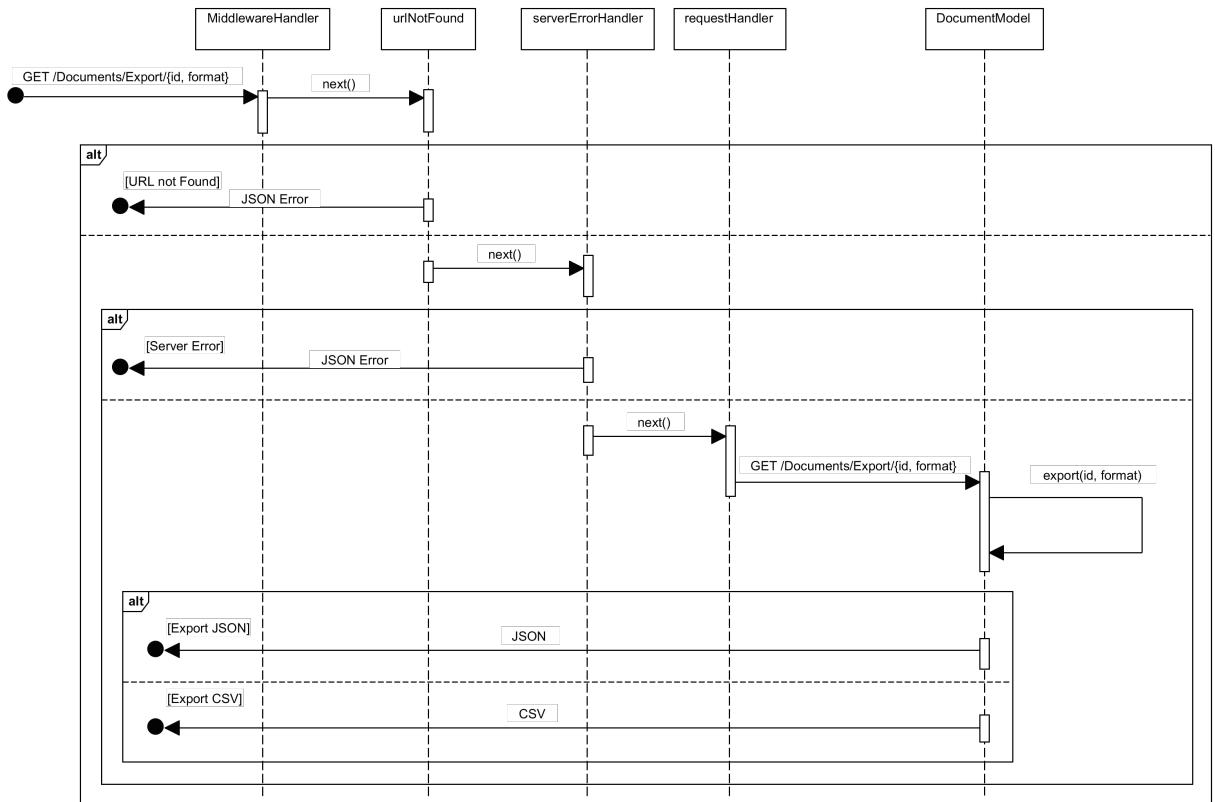


Figura 76: Diagramma di sequenza: GET /Documents/Export/{id, format}

#### Richiesta GET /Documents/retrieveDocumentDSLIS/{id}

- **Descrizione:** Ritorna il codice del Document.
- **Richiesta HTTP:** /Documents/retrieveDocumentDSLIS
- **Metodo:** GET
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Documents/retrieveDocumentDSLIS/{id}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta al `DocumentModel` che chiama il metodo `retrieveDocument(id)` che dato un id ritorna il codice relativo al Document cercato. Il `DocumentModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

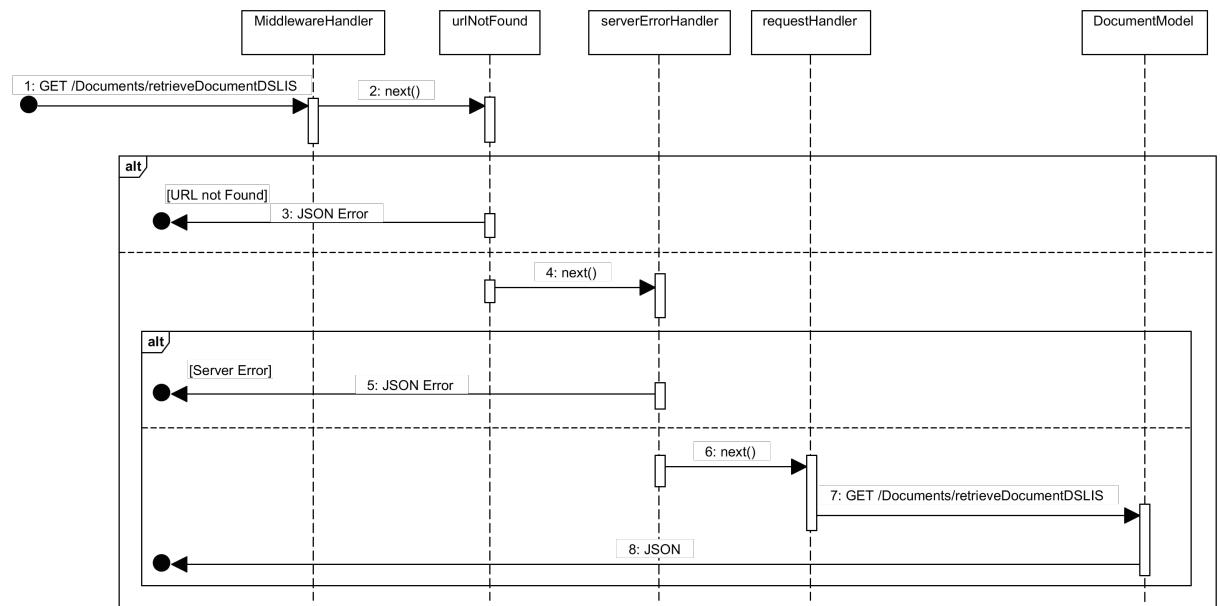


Figura 77: Diagramma di sequenza: GET /Documents/retrieveDocumentDSLIS

#### Richiesta GET /Documents/exportDocumentDSLIS

- **Descrizione:** Permette di esportare un DSLIS rappresentante un Document.
- **Richiesta HTTP:** /Documents/exportDocumentDSLIS
- **Metodo:** GET
- **Permessi:** Proprietario, Amministratore, Membro.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Documents/ExecuteDocument/{Collection}/{id}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al DocumentModel che chiama il metodo exportDSLIS(id) che dato un id permette di esportare in formato JSON il DSLIS del Document associato. Il DocumentModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

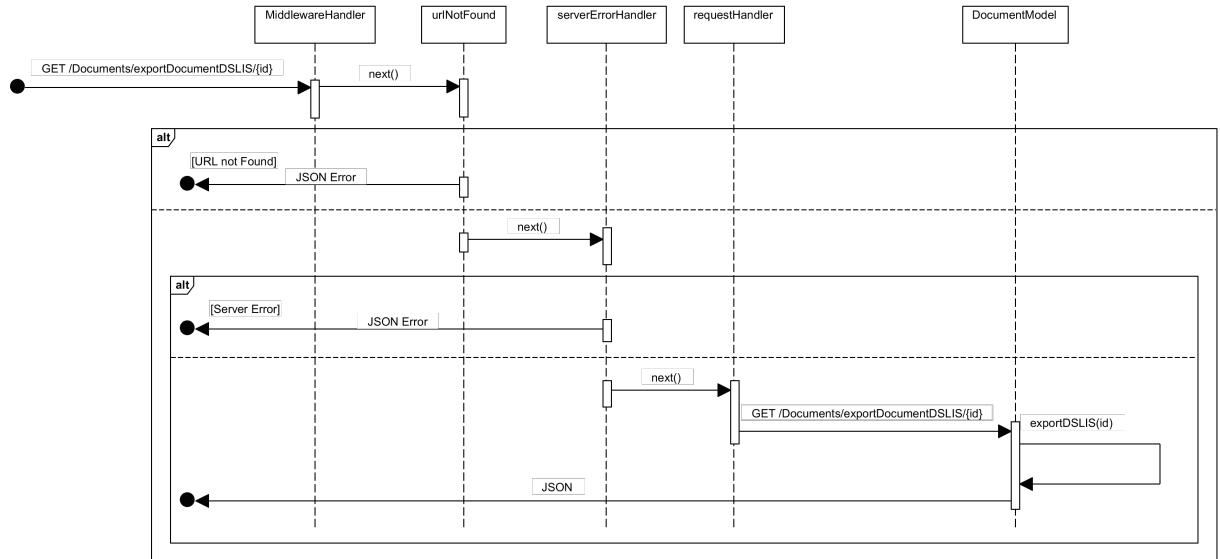


Figura 78: Diagramma di sequenza: GET /Documents/exportDocumentDSLIS

#### Richiesta POST /Documents/importDocumentDSLIS

- **Descrizione:** Permette di importare un DSLIS rappresentante un Document.
- **Richiesta HTTP:** /Documents/importDocumentDSLIS
- **Metodo:** POST
- **Permessi:** Proprietario, Amministratore, Membro.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa /Documents/importDocumentDSLIS. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al DocumentModel che chiama il metodo importDSLIS() che permette di importare in formato JSON il DSLIS relativo ad un Document. Il DocumentModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

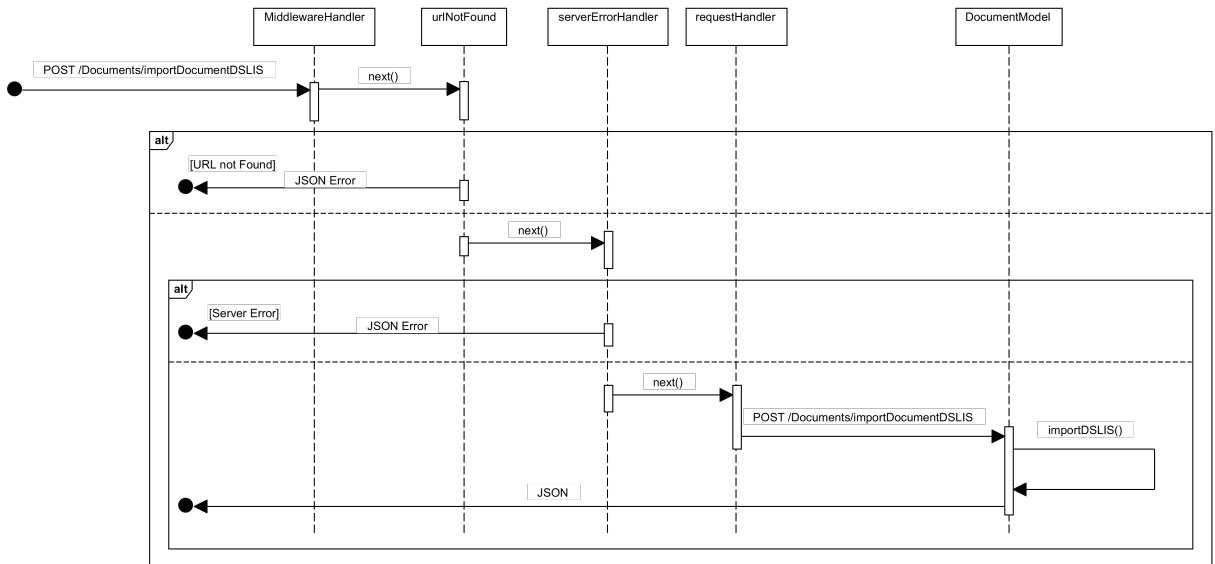


Figura 79: Diagramma di sequenza: POST /Documents/importDocumentDSLIS

### 5.2.3.7 Richieste Dashboards

#### Richiesta GET /Dashboards

- Descrizione:** Mostra la lista della Dashboard.
- Richiesta HTTP:** /Dashboards
- Metodo:** GET
- Permessi:** Utente autenticato
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Dashboards. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al DashboardModel che chiama il metodo getDashboards() che mostra la lista delle Dashboards dipendibili all'utente autenticato. Il DashboardModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

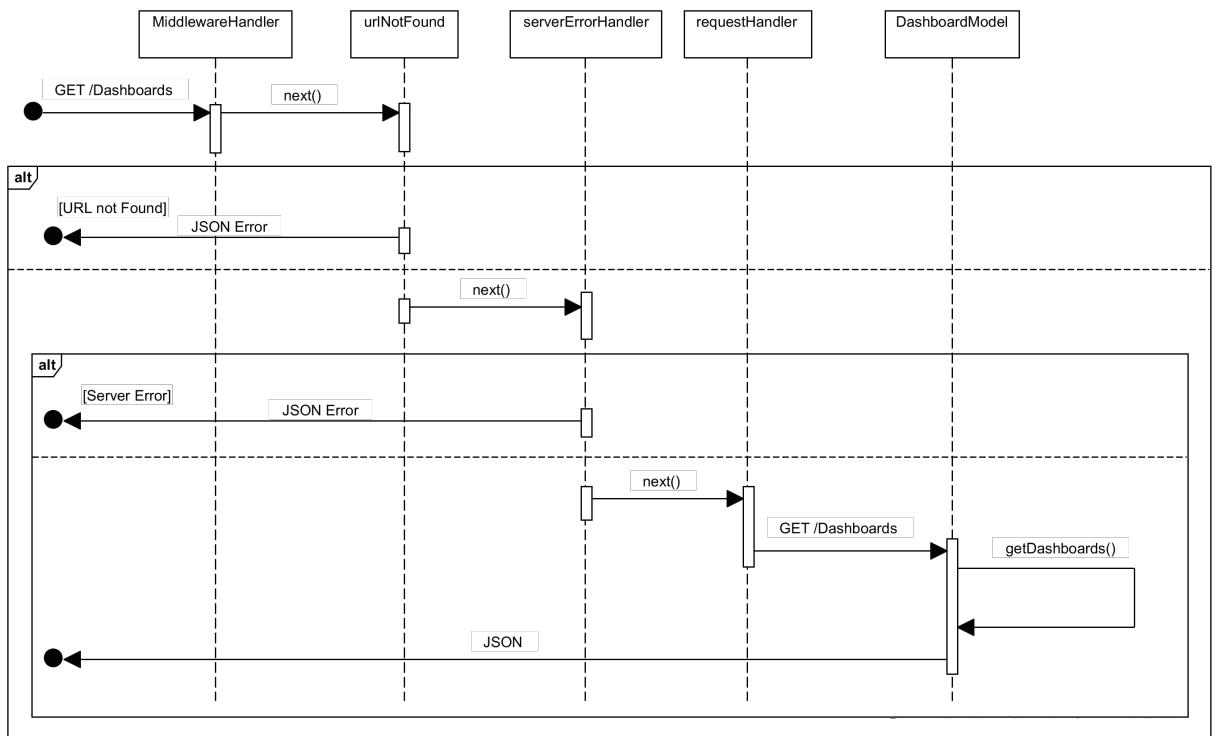


Figura 80: Diagramma di sequenza: GET /Dashboards

### Richiesta POST /Dashboards

- **Descrizione:** Creazione di una Dashboards.
- **Richiesta HTTP:** /Dashboards
- **Metodo:** POST
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa /Dashboards. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al DashboardModel che chiama il metodo createDashboard() che permette di creare una Dashboard. Il DashboardModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

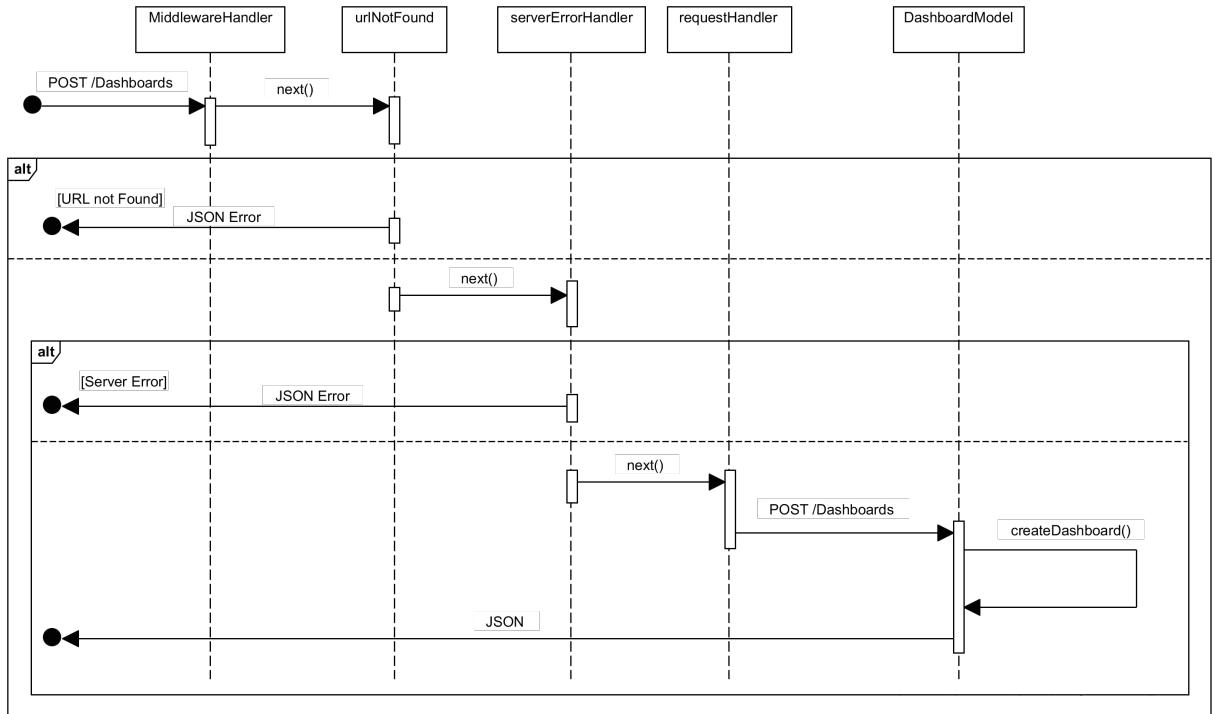


Figura 81: Diagramma di sequenza: POST /Dashboards

### Richiesta PUT /Dashboards

- **Descrizione:** Modifica di una Dashboards.
- **Richiesta HTTP:** /Dashboards
- **Metodo:** PUT
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta PUT per la risorsa /Dashboards. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al DashboardModel che chiama il metodo editDashboard() che permette di modificare una Dashboard esistente. Il DashboardModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

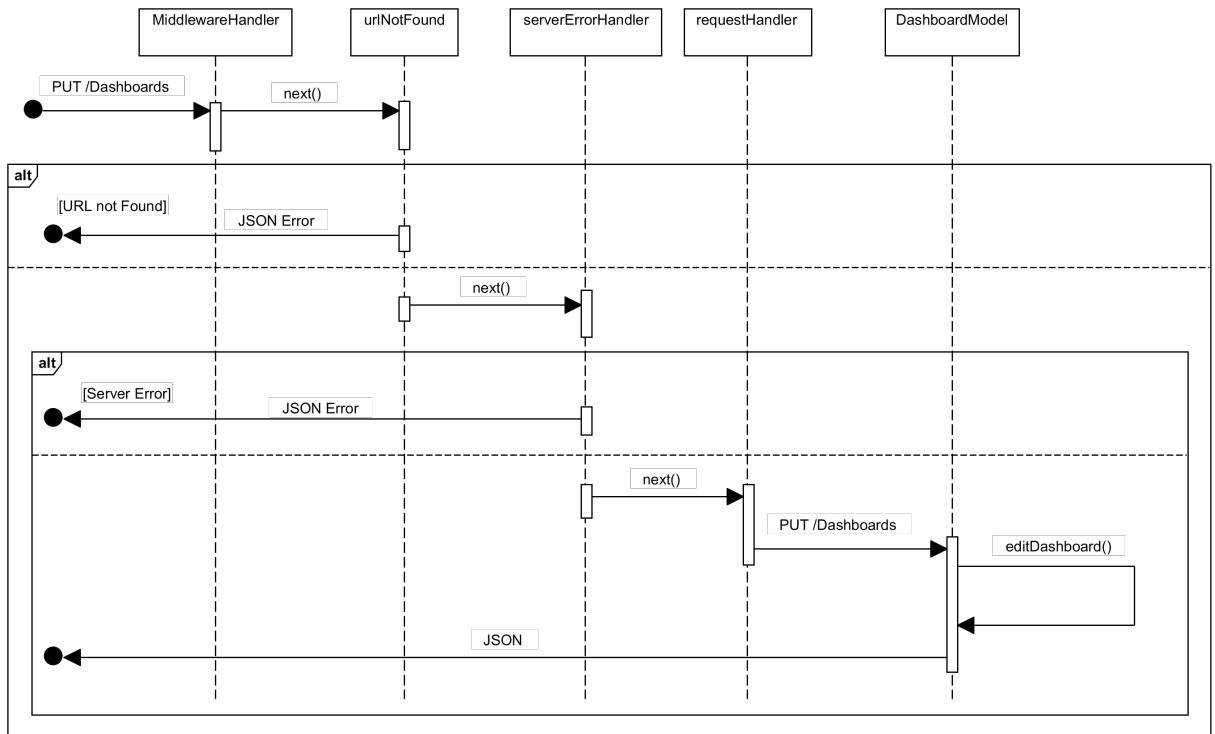
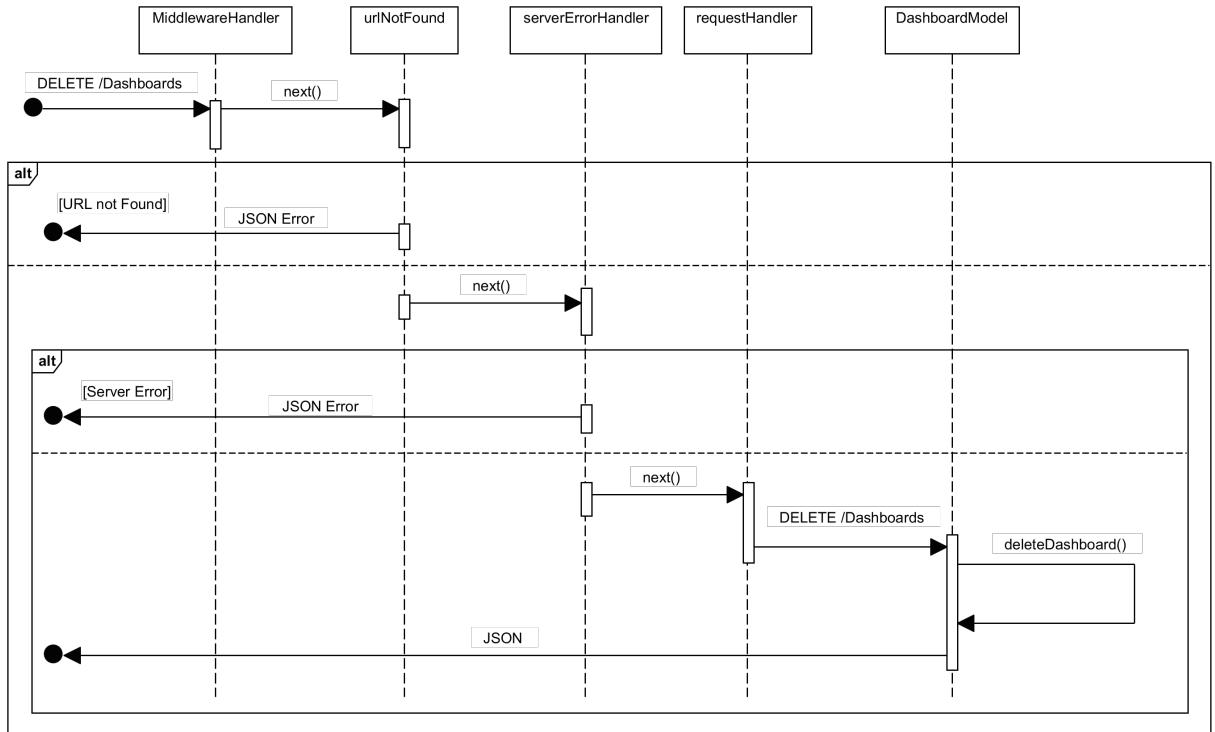


Figura 82: Diagramma di sequenza: PUT /Dashboards

#### Richiesta DELETE /Dashboards

- **Descrizione:** Eliminazione di una Dashboard mediante il identificativo.
- **Richiesta HTTP:** /Dashboards
- **Metodo:** DELETE
- **Permessi:** Utente autenticato.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta DELETE per la risorsa /Dashboards. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al DashboardModel che chiama il metodo deleteDashboard() che elimina una Dashboard esistente. Il DashboardModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.


 Figura 83: Diagramma di sequenza: `DELETE /Dashboards`

#### Richiesta `GET /Dashboards/retrieveDashboardDSLIS/{id}`

- **Descrizione:** Ritorna il codice della Dashboard
- **Richiesta HTTP:** `/Dashboards/retrieveDashboardDSLIS/{id}`
- **Metodo:** GET
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa `/Dashboards/retrieveDashboardDSLIS/{id}`. La richiesta viene passata al `MiddlewareHandler` il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta al `DashboardModel` che chiama il metodo `retrieveDashboard(id)` che ritorna il codice della Dashboard passata per id. Il `DashboardModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

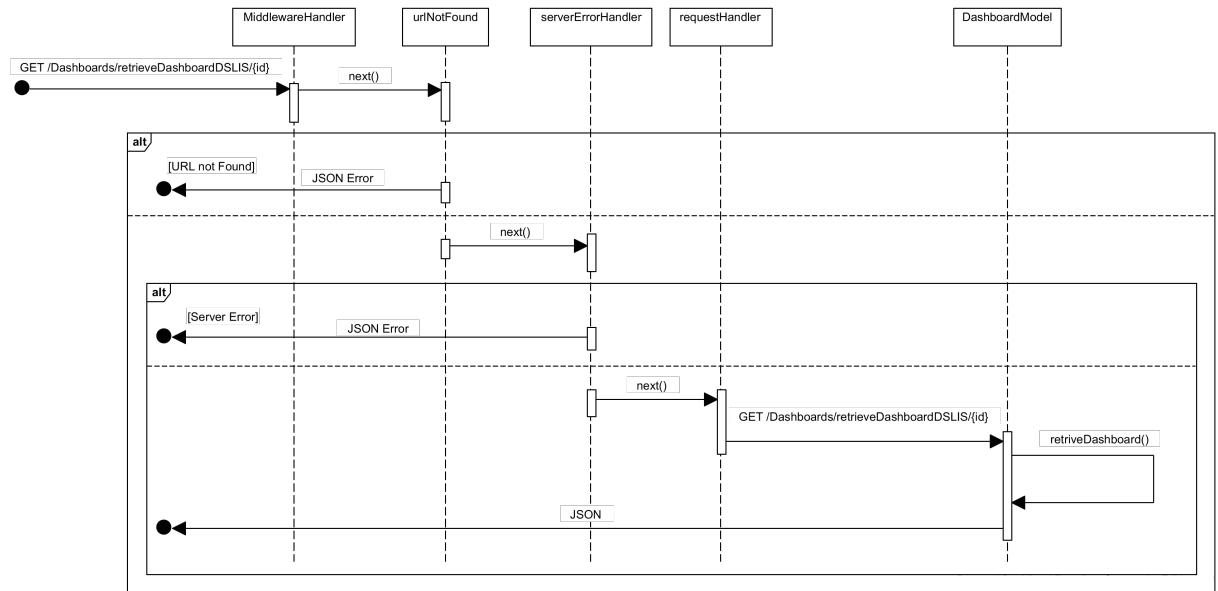


Figura 84: Diagramma di sequenza: GET /Dashboards/retrieveDashboardDSLIS/{id}

#### Richiesta GET /Dashboards/searchDashboards/{value, email}

- **Descrizione:** Ricerca di una Dashboard.
- **Richiesta HTTP:** /Dashboards/searchDashboards/{value, email}
- **Metodo:** GET
- **Permessi:** Utente autenticato.
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Dashboards/searchDashboards/{value, email}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta al `DashboardModel` che chiama il metodo `searchDashboard(value, email)` che effettua la ricerca delle Dashboards mediante un parametro, mostrando solamente quelle a cui l'utente con l'email corrispondente ha accesso. Il `DashboardModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

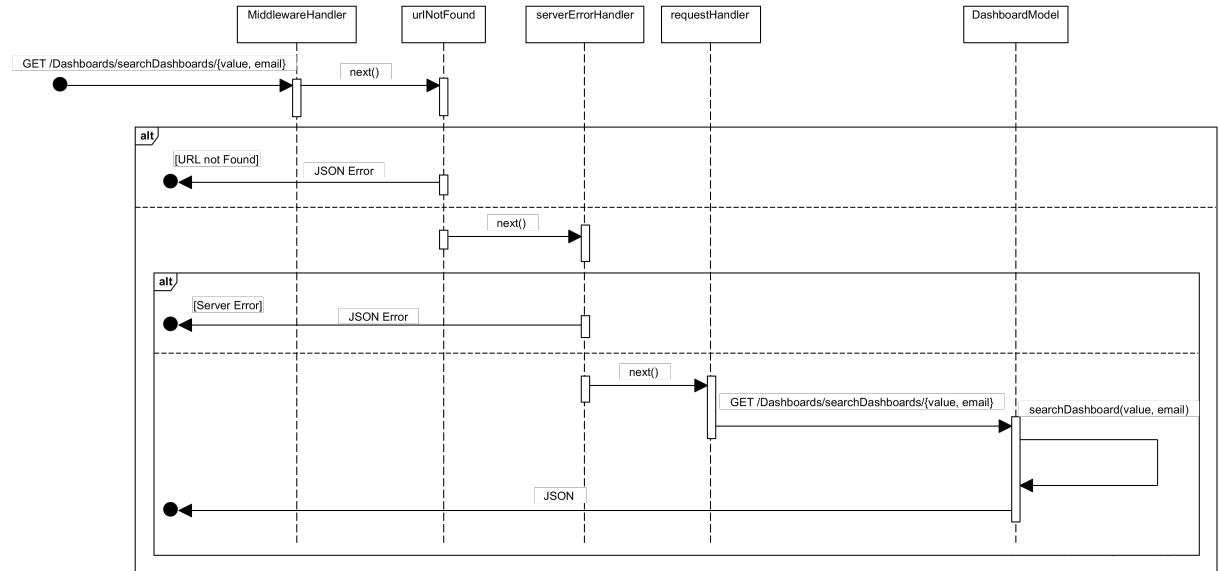


Figura 85: Diagramma di sequenza: GET /Dashboards/searchDashboards/{value, email}

#### Richiesta GET /Dashboards/ExecuteDashboard/{id}

- **Descrizione:** Ritorna l'insieme delle Row della Dashboard.
- **Richiesta HTTP:** /Dashboards/ExecuteDashboard/{id}
- **Metodo:** GET
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Dashboards/ExecuteDashboard/{id}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al DashboardModel che chiama il metodo executeDashboard(id) che ritorna l'insieme delle Row relative alla Dashboard passata per parametro. Il DashboardModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

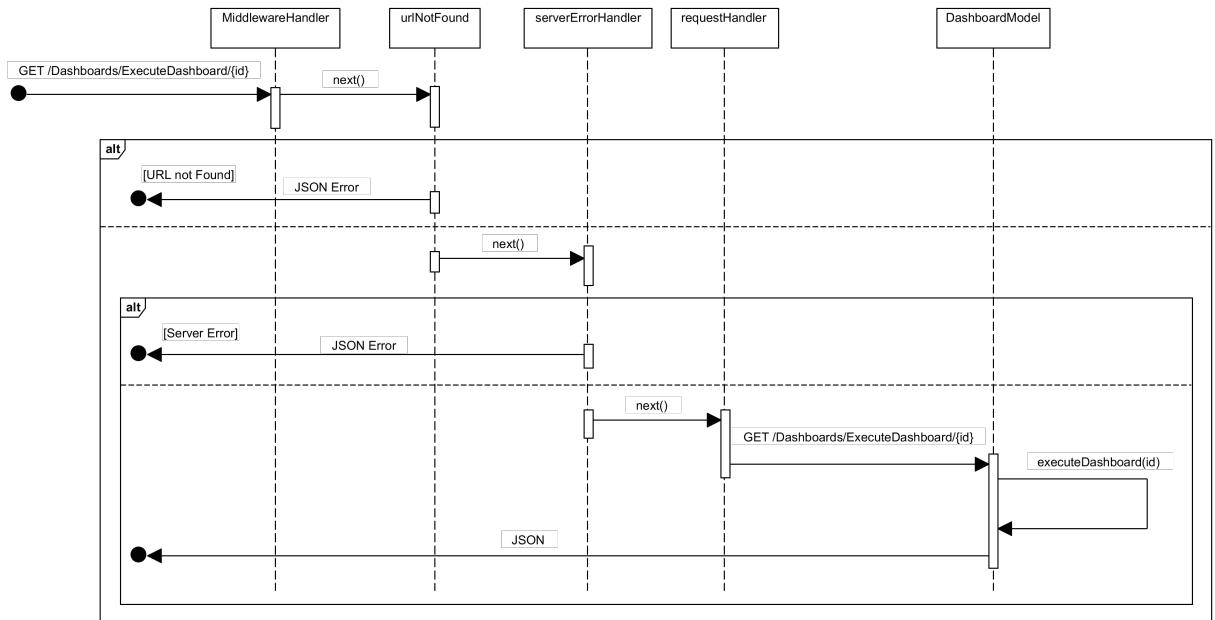


Figura 86: Diagramma di sequenza: GET /Dashboards/ExecuteDashboard/{id}

#### Richiesta GET /Dashboards/exportDashboardDSLIS/{id}

- Descrizione:** Permette di esportare un DSLIS rappresentante una Dashboard.
- Richiesta HTTP:** /Dashboards/exportDashboardDSLIS/{id}
- Metodo:** GET
- Permessi:** Proprietario, Amministratore, Membro
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Dashboards/exportDashboardDSLIS/{id}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta al `DashboardModel` che chiama il metodo `exportDSLIS(id)` che permette di esportare in formato JSON il DSLIS associato alla Dashboard passata come parametro. Il `DashboardModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

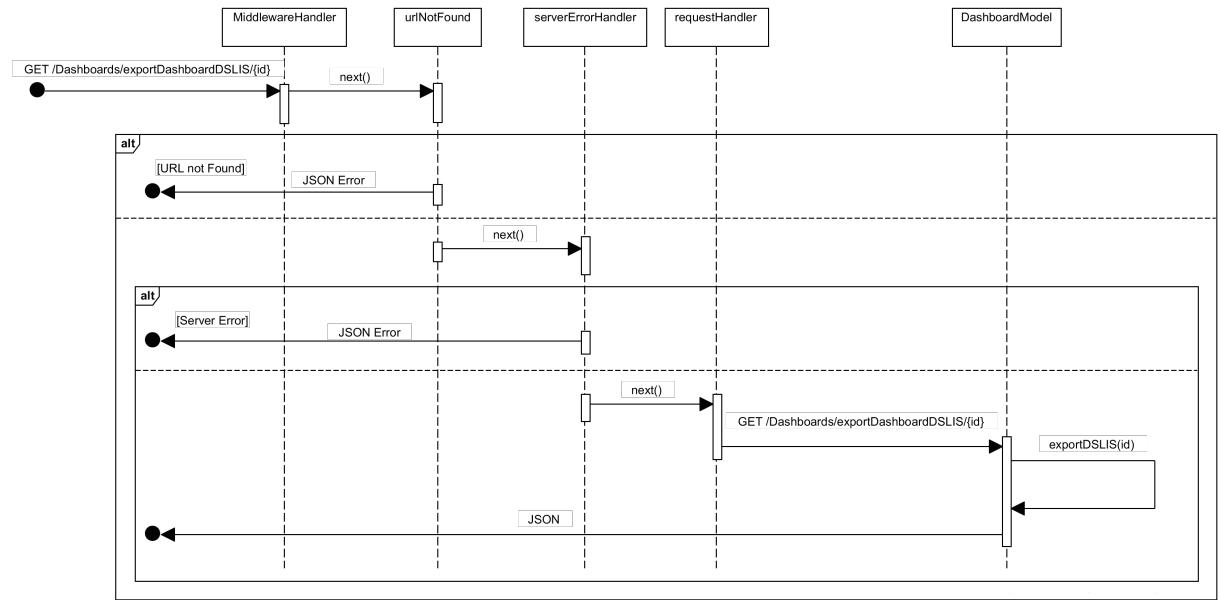


Figura 87: Diagramma di sequenza: GET /Dashboards/exportDashboardDSLIS/{id}

#### Richiesta POST /Dashboards/importDashboardDSLIS

- Descrizione:** Permette di importare un DSLIS rappresentante una Dashboard.
- Richiesta HTTP:** /Dashboards/importDashboardDSLIS
- Metodo:** GET
- Permessi:** Proprietario, Amministratore, Membro.
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Dashboards/importDashboardDSLIS/{id}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al DashboardModel che chiama il metodo importDSLIS() che permette di importare in formato JSON il DSLIS associato alla Dashboard . Il DashboardModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

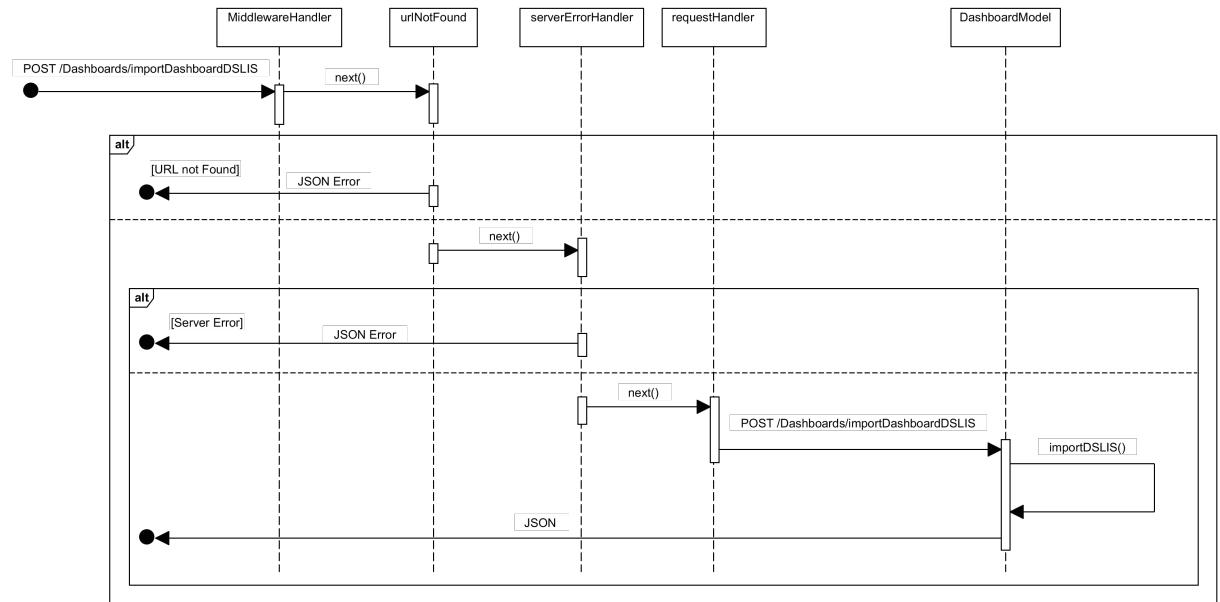


Figura 88: Diagramma di sequenza: POST /Dashboards/importDashboardDSLIS

#### 5.2.3.8 Richieste Cell

##### Richiesta GET /Cells

- **Descrizione:** Mostra la lista dei Cell.
- **Richiesta HTTP:** /Cells
- **Metodo:** GET
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Cells. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al CellModel che chiama il metodo getCells() il quale mostra la lista di tutte le Cell disponibili ad un utente. Il CellModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

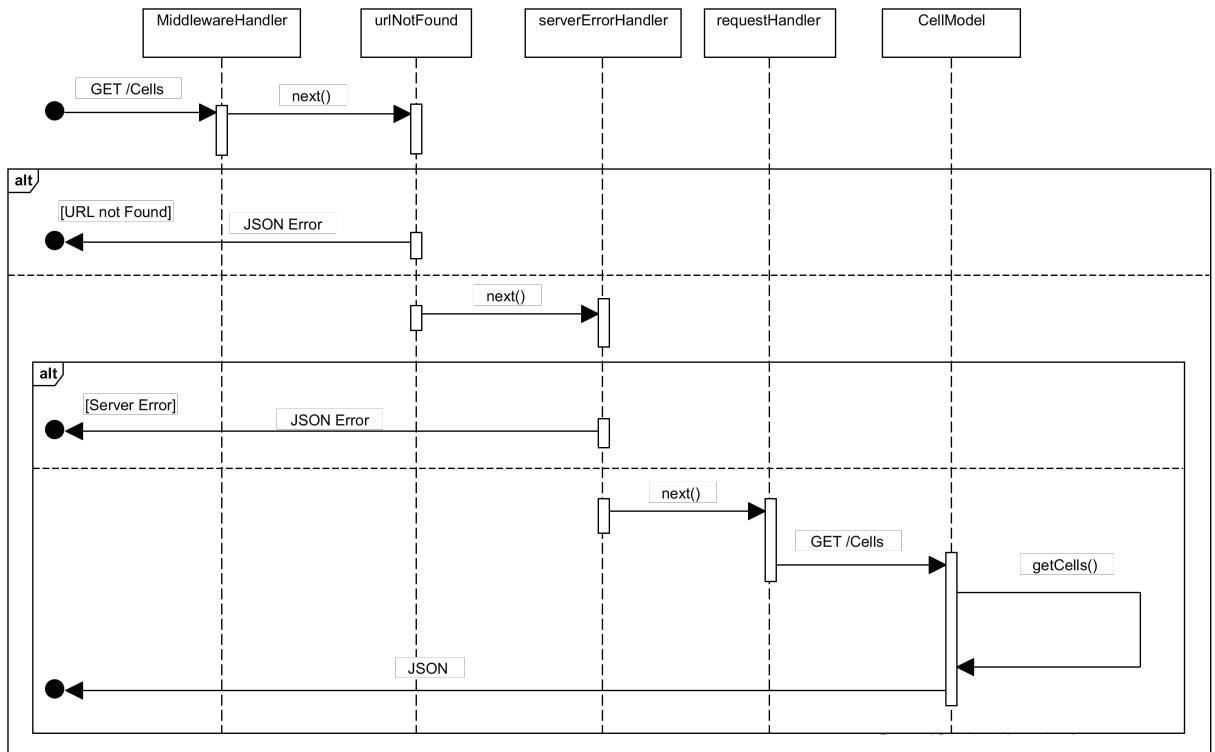


Figura 89: Diagramma di sequenza: GET /Cells

#### Richiesta POST /Cells

- **Descrizione:** Creazione di una Cell.
- **Richiesta HTTP:** /Cells
- **Metodo:** POST
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa /Cells. La richiesta viene passata al `MiddlewareHandler` il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta al `CellModel` che chiama il metodo `createCell()` il quale permette di creare un Cell. Il `CellModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

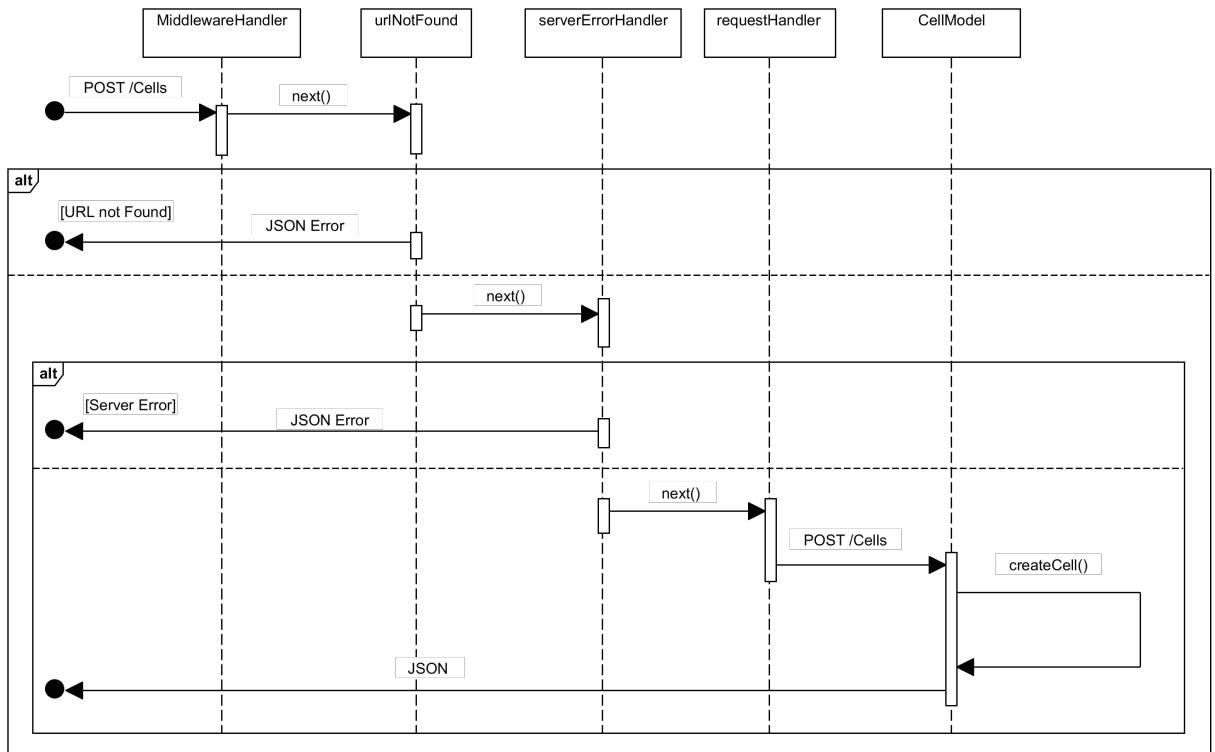


Figura 90: Diagramma di sequenza: POST /Cells

### Richiesta PUT /Cells

- **Descrizione:** Modifica di una Cell.
- **Richiesta HTTP:** /Cells
- **Metodo:** PUT
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta PUT per la risorsa /Cells. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al CellModel che chiama il metodo editCell() il quale permette di modificare una Cell esistente. Il CellModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

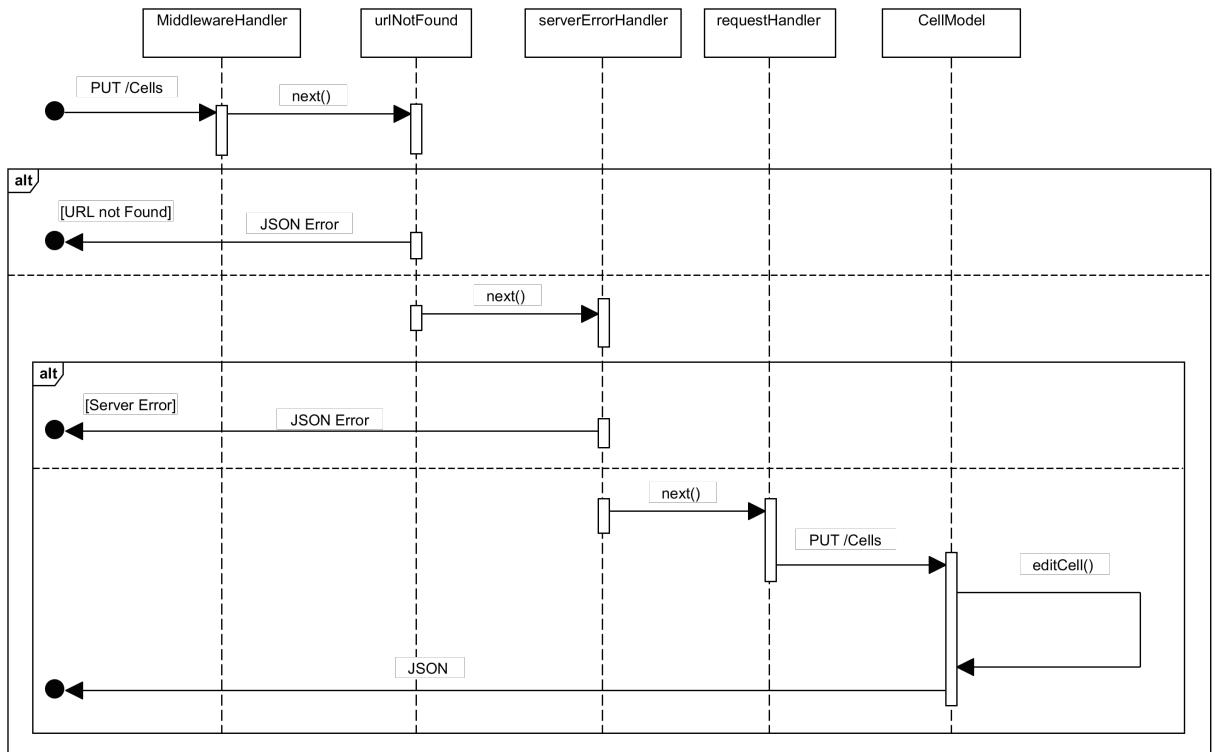
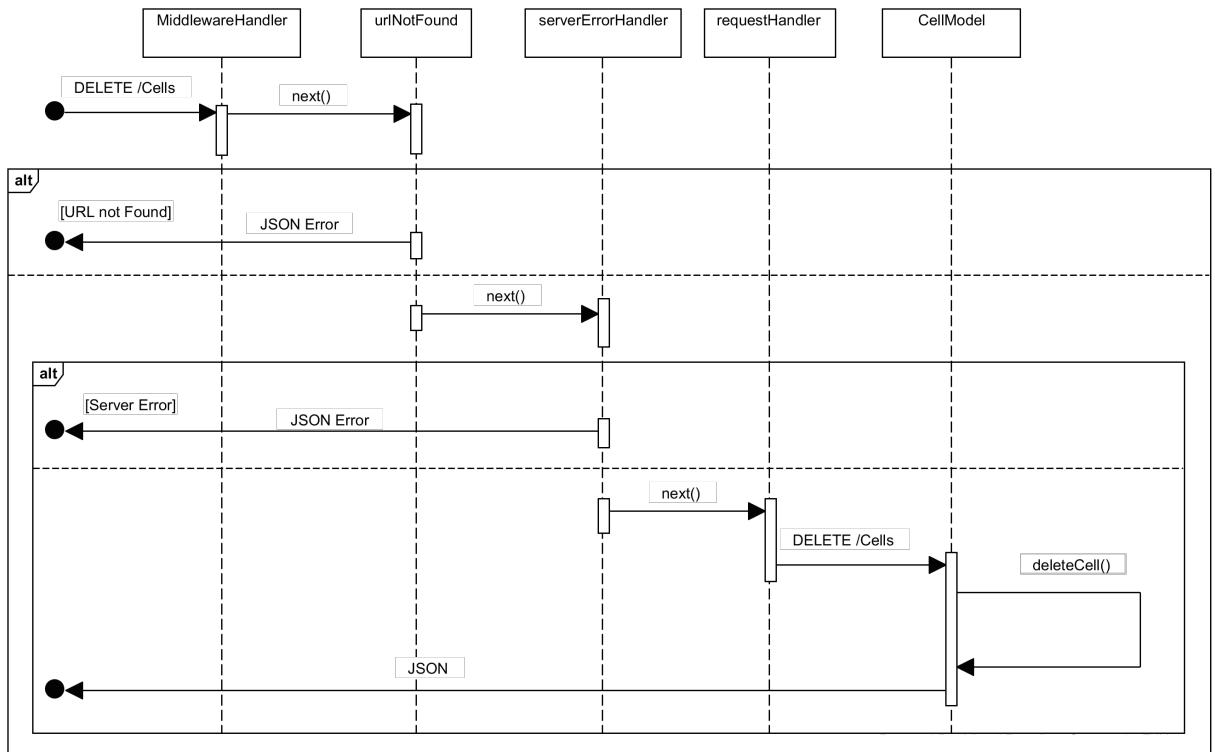


Figura 91: Diagramma di sequenza: PUT /Cells

#### Richiesta DELETE /Cells

- **Descrizione:** Eliminazione di una Cell mediante il suo identificativo.
- **Richiesta HTTP:** /Cells
- **Metodo:** DELETE
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta DELETE per la risorsa /Cells. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al CellModel che chiama il metodo deleteCell() il quale permette di eliminare una Cell esistente. Il CellModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.


 Figura 92: Diagramma di sequenza: `DELETE /Cells`

#### Richiesta GET /Cells/searchCell/{value, email}

- **Descrizione:** Ricerca di una Cell.
- **Richiesta HTTP:** `/Cells/searchCell/{value, email}`
- **Metodo:** GET
- **Permessi:** Utente autenticato
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa `/Cells/searchCell/{value, email}`. La richiesta viene passata al `MiddlewareHandler` il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta al `CellModel` che chiama il metodo `searchCell(value, email)` il quale effettua la ricerca delle Cells, a cui l'utente corrispondente all'email ha accesso, mediante un parametro. Il `CellModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

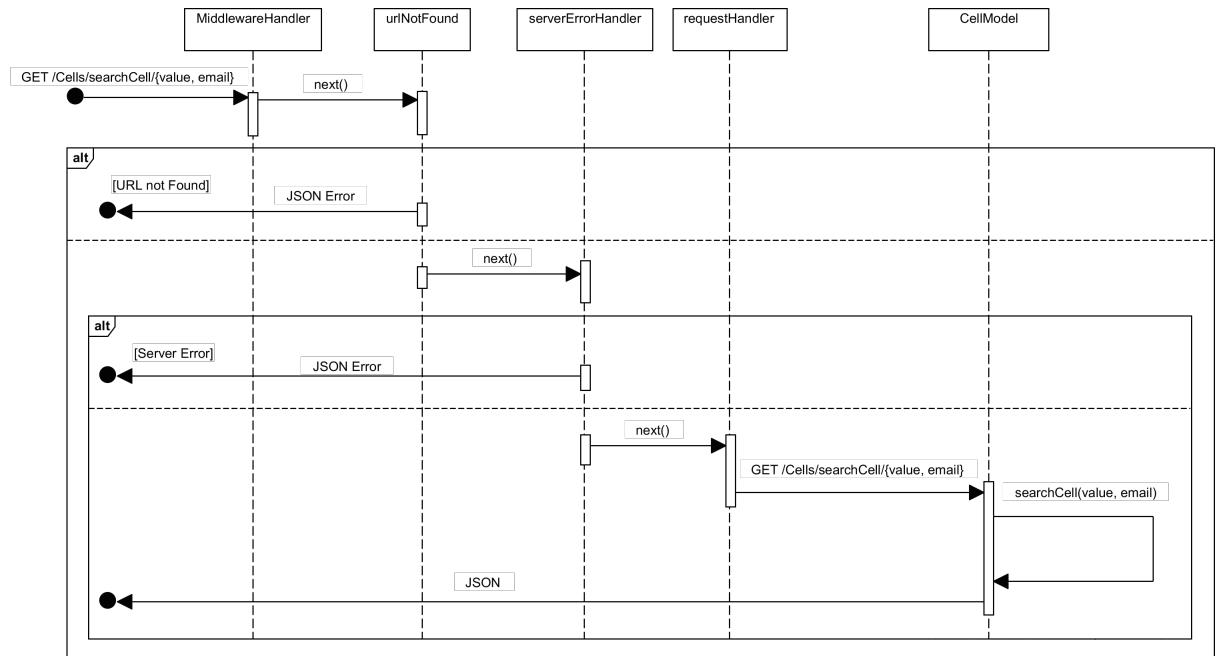


Figura 93: Diagramma di sequenza: GET /Cells/searchCell/{value, email}

**Richiesta GET /Cells/ExecuteCell/{id}**

- **Descrizione:** Esegue la Cell.
  - **Richiesta HTTP:** /Cells/ExecuteCell/{id}
  - **Metodo:** GET
  - **Permessi:** Utente autenticato
  - **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Cells/ExecuteCell/{id}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al CellModel che chiama il metodo executeCell(id) il quale esegue la Cell passata come parametro. Il CellModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

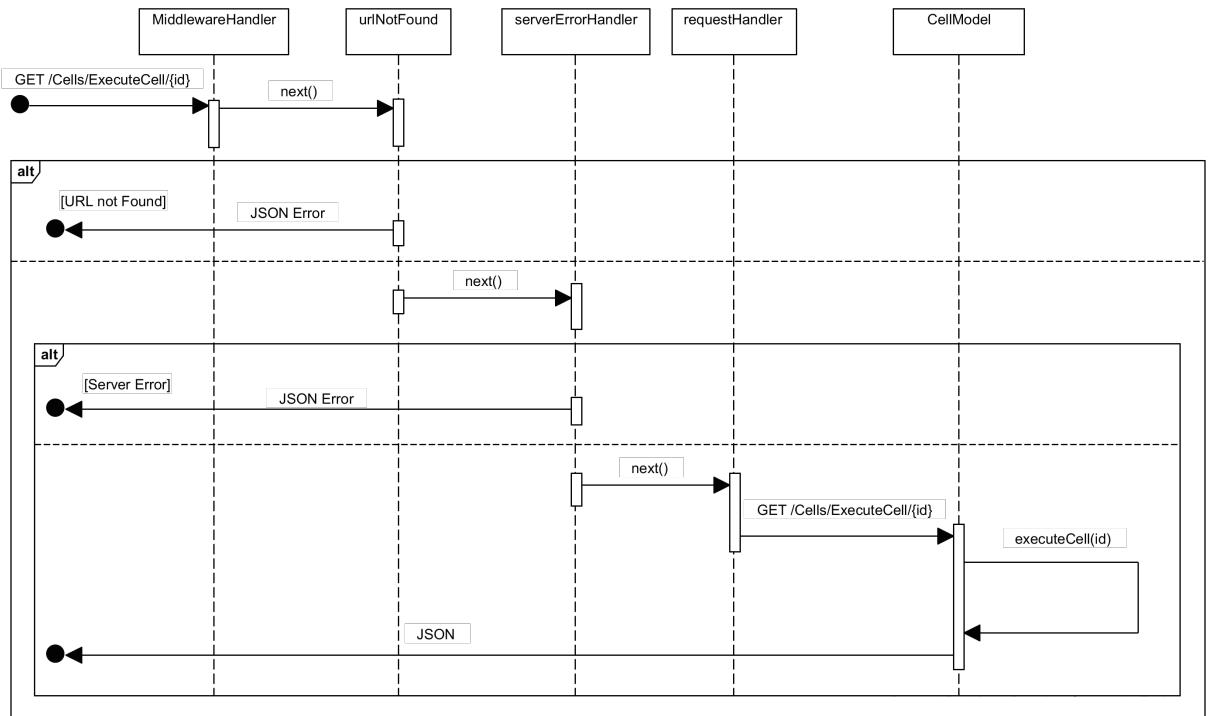


Figura 94: Diagramma di sequenza: `GET /Cells/ExecuteCell/{id}`

#### Richiesta `GET /Cells/retrieveCellDSLIS{id}`

- Descrizione:** Ritorna il codice di una Cell.
- Richiesta HTTP:** `/Cells/retrieveCellDSLIS{id}`
- Metodo:** GET
- Permessi:** Utente autenticato
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa `/Cells/retrieveCellDSLIS{id}`. La richiesta viene passata al `MiddlewareHandler` il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta al `CellModel` che chiama il metodo `retriveCell(id)` il quale ritorna il codice della Cell richiesta. Il `CellModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

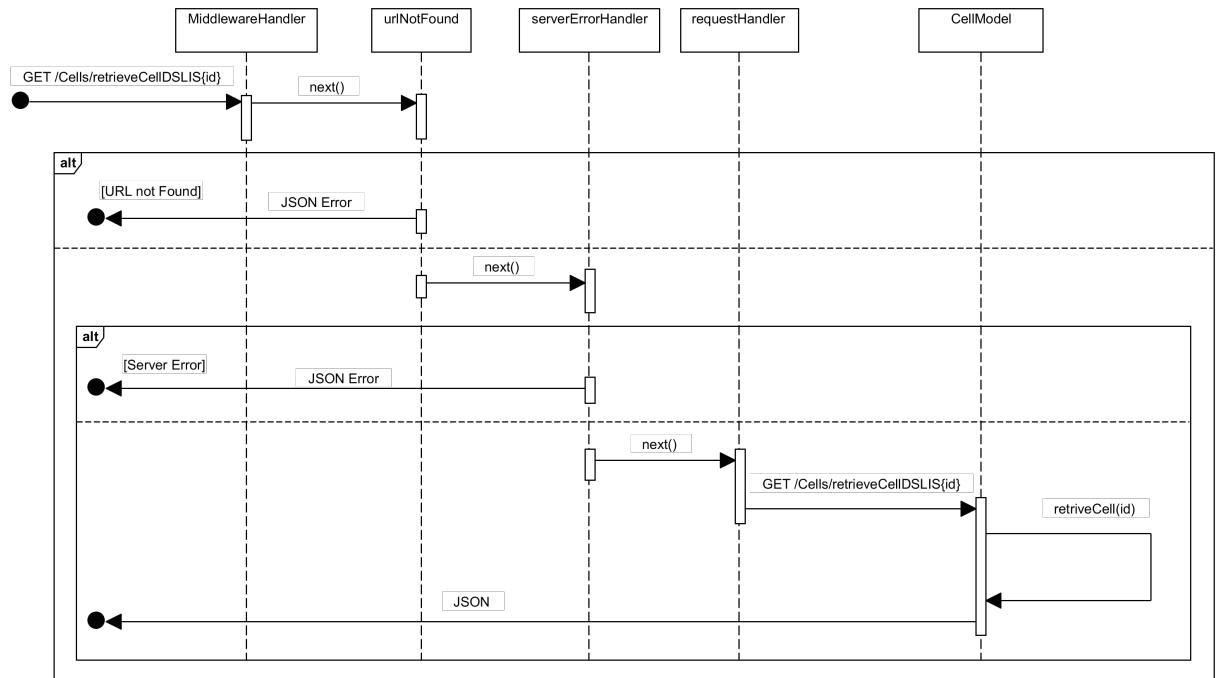


Figura 95: Diagramma di sequenza: GET /Cells/GET /Cells/retrieveCellDSLIS{id}

#### Richiesta GET /Cells/exportCellDSLIS/{id}

- Descrizione:** Permette di esportare un DSLIS rappresentante una Cell.
- Richiesta HTTP:** /Cells/exportCellDSLIS/{id}
- Metodo:** GET
- Permessi:** Proprietario, Amministratore, Membro
- Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta GET per la risorsa /Cells/exportCellDSLIS/{id}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility `urlNotFound` e `serverErrorHandler` si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il `requestHandler` passa la richiesta al `CellModel` che chiama il metodo `exportDSLIS(id)` il quale permette l'esportazione in formato JSON del DSLIS associato alla Cell passata per parametro. Il `CellModel` si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

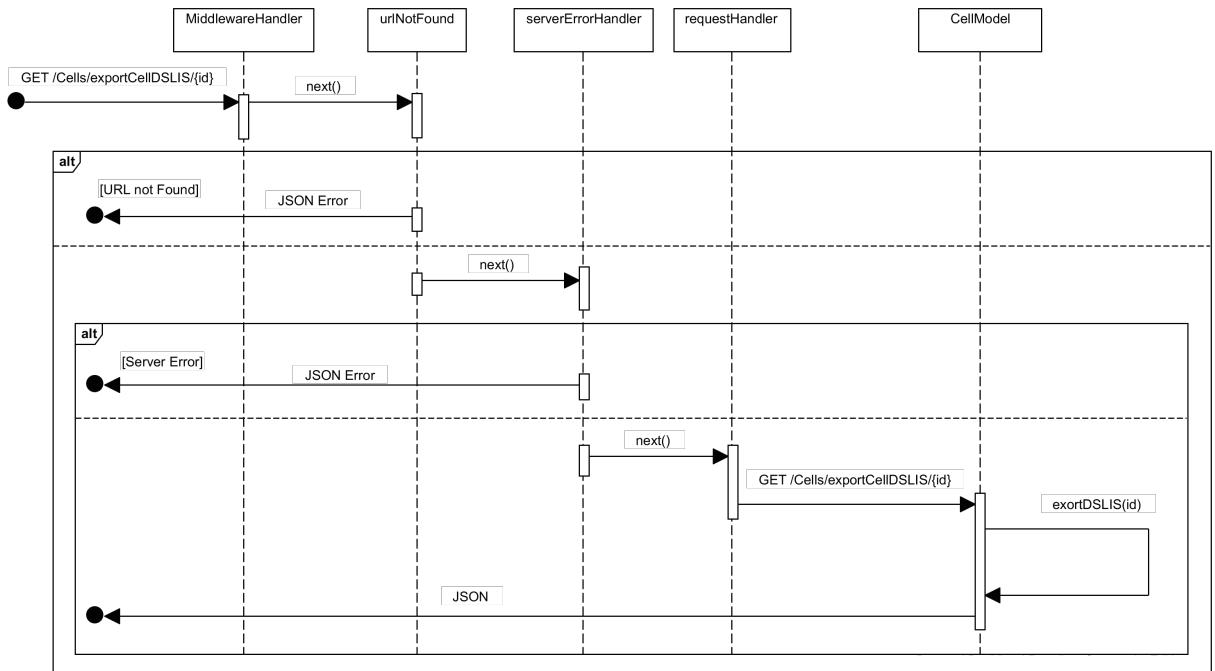


Figura 96: Diagramma di sequenza: GET /Cells/exportCellDSLIS/{id}

#### Richiesta POST /Cells/importCellDSLIS

- **Descrizione:** Permette di importare un DSLIS rappresentante una Cell.
- **Richiesta HTTP:** /Cells/importCellDSLIS
- **Metodo:** POST
- **Permessi:** Proprietario, Amministratore, Membro
- **Scenario:** Il diagramma di sequenza mostra lo scenario di una richiesta POST per la risorsa /Cells/importCellDSLIS/{id}. La richiesta viene passata al MiddlewareHandler il quale si occupa di escludere due tipi di errore. Le successive classi della Chain of Responsibility urlNotFound e serverErrorHandler si occupano rispettivamente di controllare se vi sono errori di indirizzo non trovato e errori lato server, ritornando eventualmente un errore in formato JSON. Se non vi sono errori il requestHandler passa la richiesta al CellModel che chiama il metodo importDSLIS(id) il quale permette l'importazione in formato JSON del DSLIS associato ad una Cell. Il CellModel si occupa di ritornare un JSON informando il client sull'esito dell'operazione.

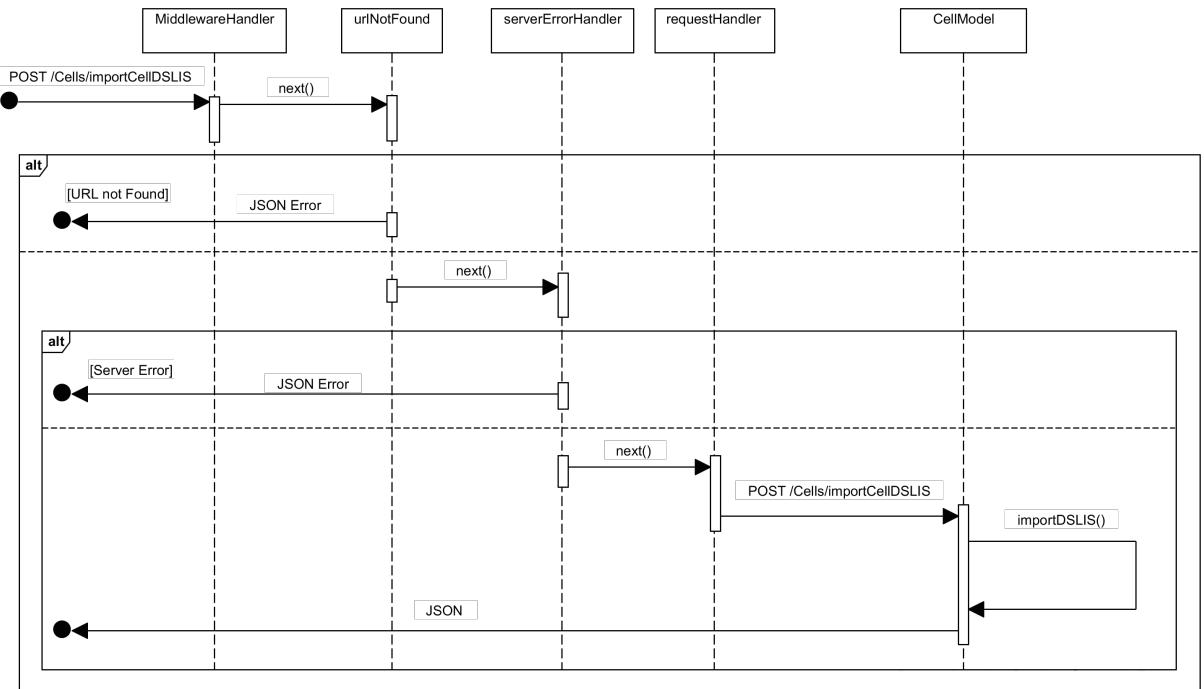


Figura 97: Diagramma di sequenza: POST /Cells/importCellDSLIS

## 6 Front-end

### 6.1 Descrizione packages e classi

#### 6.1.1 Front-end

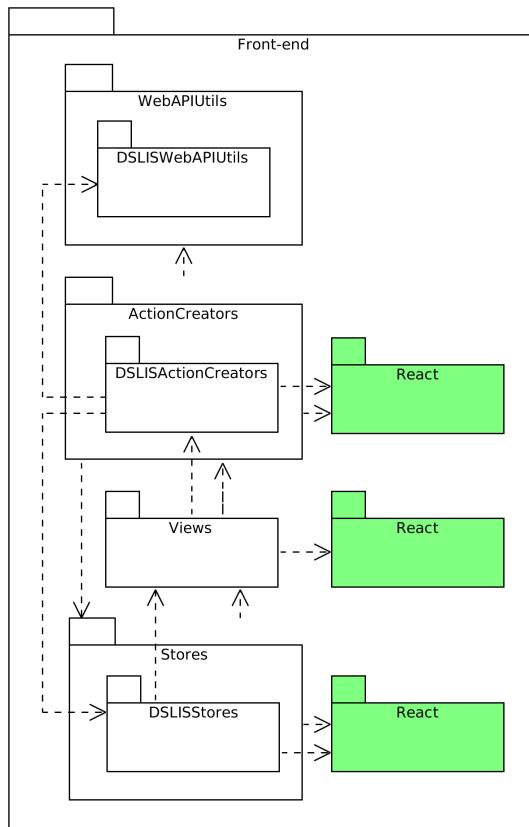


Figura 98: Diagramma dei package del Front-end

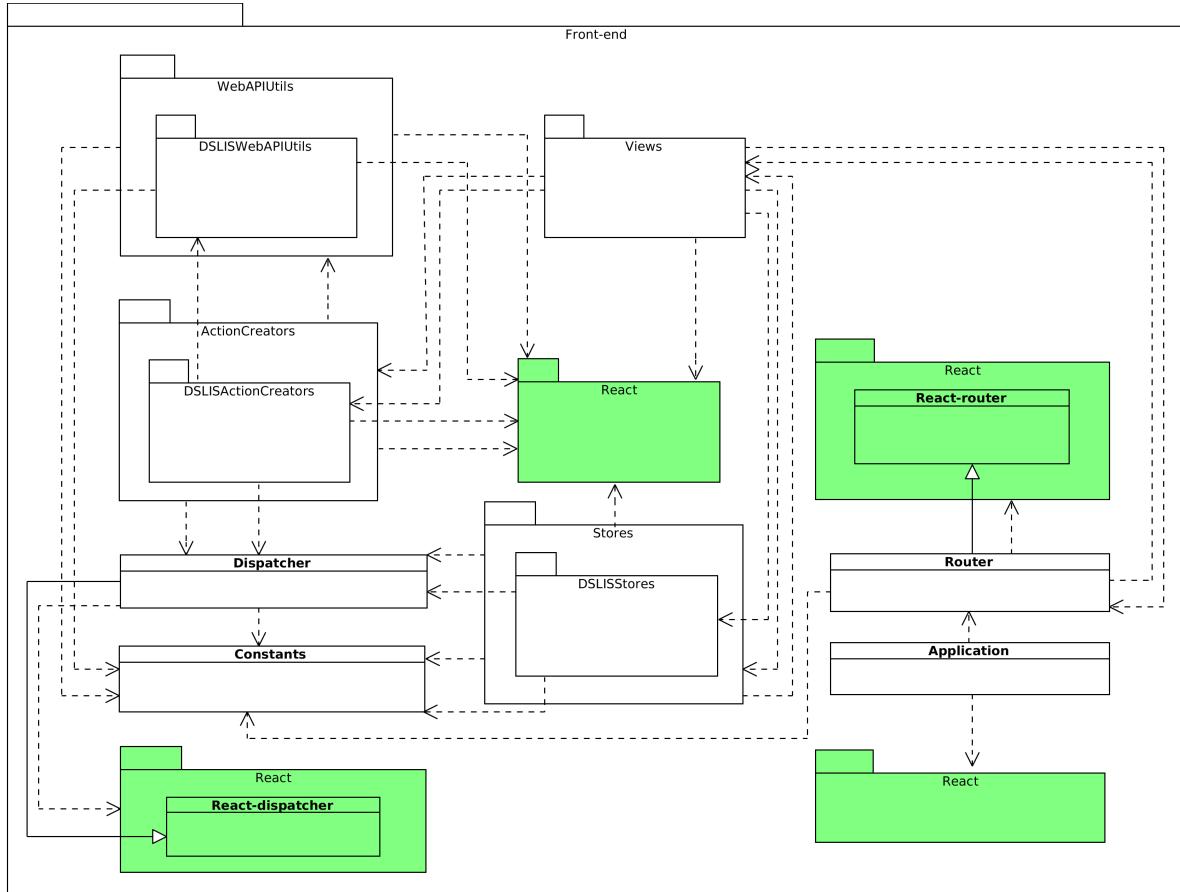


Figura 99: Diagramma delle classi del Front-end

#### 6.1.1.1 Informazioni sul package

##### Descrizione

Package che racchiude tutte le componenti del Front-end. Comprende tutte le classi che rappresentano la parte client dell'applicativo.

Le componenti sono organizzate secondo il design pattern Flux.

##### Package contenuti

- ActionCreators
- WebAPIUtils
- Stores
- Views

## Framework esterni

- React

### 6.1.1.2 Classi

#### Dispatcher

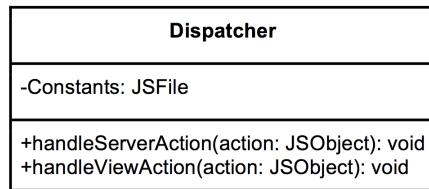


Figura 100: Diagramma della classe Dispatcher

#### Descrizione

Classe che rappresenta il componente Dispatcher, ovvero colui che ha il compito di inoltrare le Action alle relative Store.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

#### Utilizzo

Viene utilizzato per trasferire le Action, create dalle classi contenute nel package Front-end::Actions, alle classi rappresentative della Store, contenute nel package Front-end::Stores. Questa classe fornisce un metodo, alle classi ActionCreator, per inviare le Action alle relative Store. Inoltre, fornisce un metodo, alle classi Store, per ricevere e registrare tali Action.

#### Classi ereditate

- React::React-dispatcher

#### Relazioni con altre classi

- Front-end::Constants

#### Attributi

**-Constants : JSFile**

Questo attributo rappresenta un riferimento al file constants.js.

## Metodi

**+handleServerAction(action: JSObject) : void**

Questo metodo si occupa di preparare una action riguardante la risposta del back-end e di inoltrarla, utilizzando il metodo `dispatch()` del framework React;

**action : JSObject**

Questo parametro rappresenta un oggetto javascript contenente le informazioni di una action;

**+handleViewAction(action: JSObject) : void**

Questo metodo si occupa di preparare una action riguardante una richiesta del client e di inoltrarla, utilizzando il metodo `dispatch()` del framework React;

**action : JSObject**

Questo parametro rappresenta un oggetto javascript contenente le informazioni di una action.

## Application

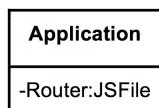


Figura 101: Diagramma della classe Application

## Descrizione

Classe che si occupa di lanciare l'applicazione. Viene derivata dal framework React.

## Utilizzo

Viene utilizzata per lanciare l'applicazione e mostrare una pagina specifica. Essa istanzia ed utilizza il Router, il quale si occupa di mostrare la pagina iniziale.

## Relazioni con altre classi

- Front-end::Router

## Attributi

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js.

## Metodi

Assenti, in quanto questa classe utilizza il metodo `render()` del framework React per l'inizializzazione dell'applicazione.

## Router



Figura 102: Diagramma della classe Router

## Descrizione

Classe che contiene tutti i percorsi dell'applicazione ed i metodi per raggiungerli.  
Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare le diverse pagine dell'applicazione ed associarle ad uno specifico URL.

### Classi ereditate

- React::React-router

### Relazioni con altre classi

- Front-end::Views::RegistrazioneView
- Front-end::Views::LoginView
- Front-end::Views::RecuperoPasswordView
- Front-end::Views::GestioneDSLISView
- Front-end::Views::GestioneDatabaseEsterniView
- Front-end::Views::ModificaPermessiAccessoDatabaseEsterniView
- Front-end::Views::RegistrazioneCollaboratoreView
- Front-end::Views::CreazioneDSLISView
- Front-end::Views::ModificaDSLISView
- Front-end::Views::DashboardView
- Front-end::Views::ConfermaRegistrazioneView
- Front-end::Views::GesionePermessiDSLISView
- Front-end::Views::ImpostazioniApplicazioneView
- Front-end::Views::GestioneUtentiAziendaView
- Front-end::Views::ModificaDashboardPredefinitaView
- Front-end::Views::ModificaPasswordView
- Front-end::Views::VisualizzazioneDSLISView
- Front-end::Views::ModificaDatiAnagraficiView
- Front-end::Views::GestioneDatiAziendeView
- Front-end::Views::SuperAmministratoreDashboardView
- Front-end::Views::GestioneDatiUtenteView
- Front-end::Views::DocumentView
- Front-end::Views::DSLISListView

- Front-end::Views::CollectionView
- Front-end::Views::CellView

### Attributi

**-registrazioneView : JSFile**

Questo attributo rappresenta un riferimento al file RegistrationView.js;

**-loginView : JSFile**

Questo attributo rappresenta un riferimento al file LoginView.js;

**-recuperoPasswordView : JSFile**

Questo attributo rappresenta un riferimento al file RecuperoPasswordView.js;

**-gestioneDSLISView : JSFile**

Questo attributo rappresenta un riferimento al file GestioneDSLISView.js;

**-gestioneDatabaseEsterniView : JSFile**

Questo attributo rappresenta un riferimento al file GestioneDatabaseEsterniView.js;

**-modificaPermessiAccessoDatabaseEsterniView : JSFile**

Questo attributo rappresenta un riferimento al file ModificaPermessiAccessoDatabaseEsterniView.js;

**-registrazioneCollaboratoreView : JSFile**

Questo attributo rappresenta un riferimento al file RegistrationCollaboratoreView.js;

**-creazioneDSLISView : JSFile**

Questo attributo rappresenta un riferimento al file CreazioneDSLISView.js;

**-modificaDSLISView : JSFile**

Questo attributo rappresenta un riferimento al file ModificaDSLISView.js;

**-dashboardView : JSFile**

Questo attributo rappresenta un riferimento al file DashboardView.js;

**-confermaRegistrazioneView : JSFile**

Questo attributo rappresenta un riferimento al file ConfermaRegistrazioneView.js;

**-gestionePermessiDSLISView : JSFile**

Questo attributo rappresenta un riferimento al file GestionePermessiDSLISView.js;

**-impostazioniApplicazioneView : JSFile**

Questo attributo rappresenta un riferimento al file ImpostazioniApplicazioneView.js;

**-gestioneUtentiAziendaView : JSFile**

Questo attributo rappresenta un riferimento al file GestioneUtentiAziendaView.js;

**-modificaDashboardPredefinitaView : JSFile**

Questo attributo rappresenta un riferimento al file ModificaDashboardPredefinitaView.js;

**-modificaPasswordView : JSFile**

Questo attributo rappresenta un riferimento al file ModificaPasswordView.js;

**-visualizzazioneDSLISView : JSFile**

Questo attributo rappresenta un riferimento al file VisualizzazioneDSLISView.js;

**-modificaDatiAnagraficiView : JSFile**

Questo attributo rappresenta un riferimento al file ModificaDatiAnagraficiView.js;

**-gestioneDatiAziendeView : JSFile**

Questo attributo rappresenta un riferimento al file GestioneDatiAziendeView.js;

**-superAmministratoreDashboardView : JSFile**

Questo attributo rappresenta un riferimento al file SuperAmministratoreDashboardView.js;

**-gestioneDatiUtenteView : JSFile**

Questo attributo rappresenta un riferimento al file GestioneDatiUtenteView.js;

**-documentView : JSFile**

Questo attributo rappresenta un riferimento al file DocumentView.js;

**-DSLISListView : JSFile**

Questo attributo rappresenta un riferimento al file DSLISListView.js;

**-collectionView : JSFile**

Questo attributo rappresenta un riferimento al file CollectionView.js;

**-cellView : JSFile**

Questo attributo rappresenta un riferimento al file CellView.js.

## Metodi

**+getRegistrazioneView() : JSFile**

Questo metodo si occupa di ritornare l'attributo registrazioneView;

**+getLoginView() : JSFile**

Questo metodo si occupa di ritornare l'attributo loginView;

**+getRecuperoPasswordView() : JSFile**

Questo metodo si occupa di ritornare l'attributo recuperoPasswordView;

**+getGestioneDSLISView() : JSFile**

Questo metodo si occupa di ritornare l'attributo gestioneDSLISView;

**+getGestioneDatabaseEsterniView() : JSFile**

Questo metodo si occupa di ritornare l'attributo gestioneDatabaseEsterniView;

**+getModificaPermessiAccessoDatabaseEsterniView() : JSFile**

Questo metodo si occupa di ritornare l'attributo modificaPermessiAccessoDatabaseEsterniView;

**+registrazioneCollaboratoreView() : JSFile**

Questo metodo si occupa di ritornare l'attributo registrazioneCollaboratoreView;

**+getCreazioneDSLISView() : JSFile**

Questo metodo si occupa di ritornare l'attributo creazioneDSLISView;

**+getModificaDSLISView() : JSFile**

Questo metodo si occupa di ritornare l'attributo modificaDSLISView;

**+getDashboardView() : JSFile**

Questo metodo si occupa di ritornare l'attributo dashboardView;

```
+getConfermaRegistrazioneView() : JSFile
Questo metodo si occupa di ritornare l'attributo confermaRegistrazioneView;

+getGestionePermessiDSLISView() : JSFile
Questo metodo si occupa di ritornare l'attributo gestionePermessiDSLISView;

+getImpostazioniApplicazioneView() : JSFile
Questo metodo si occupa di ritornare l'attributo impostazioniApplicazioneView;

+gestioneUtentiAziendaView() : JSFile
Questo metodo si occupa di ritornare l'attributo gestioneUtentiAziendaView;

+getModificaDashboardPredefinitaView() : JSFile
Questo metodo si occupa di ritornare l'attributo modificaDashboardPredefinitaView;

+getModificaPasswordView() : JSFile
Questo metodo si occupa di ritornare l'attributo modificaPasswordView;

+getVisualizzazioneDSLISView() : JSFile
Questo metodo si occupa di ritornare l'attributo visualizzazioneDSLISView;

+getModificaDatiAnagraficiView() : JSFile
Questo metodo si occupa di ritornare l'attributo modificaDatiAnagraficiView;

+getGestioneDatiAziendeView() : JSFile
Questo metodo si occupa di ritornare l'attributo gestioneDatiAziendeView;

+getSuperAmministratoreDashboardView() : JSFile
Questo metodo si occupa di ritornare l'attributo superAmministratoreDashboardView;

+getGestioneDatiUtenteView() : JSFile
Questo metodo si occupa di ritornare l'attributo gestioneDatiUtenteView;

+getDocumentView() : JSFile
Questo metodo si occupa di ritornare l'attributo documentView;

+getDSLISListView() : JSFile
Questo metodo si occupa di ritornare l'attributo DSLISListView;

+getCollectionView() : JSFile
Questo metodo si occupa di ritornare l'attributo collectionView;

+getCellView() : JSFile
Questo metodo si occupa di ritornare l'attributo cellView.
```

## Constants

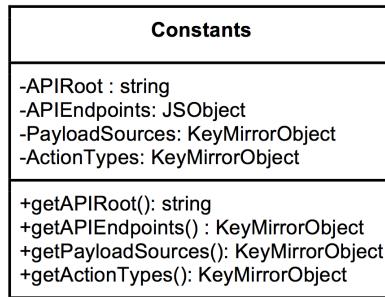


Figura 103: Diagramma della classe Constants

### Descrizione

Classe che contiene le costanti utilizzate all'interno dell'applicazione. Si utilizza la componente esterna keymirror di React per creare oggetti rappresentanti delle enumerazioni.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Classe che fornisce le costanti riguardanti i punti di arrivo delle API ed i tipi di action che si possono eseguire all'interno dell'applicazione.

### Attributi

#### **-APIRoot : string**

Questo attributo rappresenta l'indirizzo root delle API;

#### **-APIEndpoints : JSObject**

Questo attributo rappresenta un oggetto javascript contenente i punti di arrivo delle API;

#### **-PayloadSources : KeyMirrorObject**

Questo attributo rappresenta un oggetto di tipo keymirror, ovvero un'enumerazione contenente i nomi delle sorgenti di provenienza delle action;

#### **-ActionTypes : KeyMirrorObject**

Questo attributo rappresenta un oggetto di tipo keymirror, ovvero un'enumerazione contenente i nomi delle tipologie di action.

### Metodi

#### **+getAPIRoot() : string**

Questo metodo si occupa di ritornare l'attributo APIRoot;

#### **+getAPIEndpoints() : KeyMirrorObject**

Questo metodo si occupa di ritornare l'attributo APIEndpoints;

#### **+getPayloadSources() : KeyMirrorObject**

Questo metodo si occupa di ritornare l'attributo PayloadSources;

```
+getActionTypes() : KeyMirrorObject
```

Questo metodo si occupa di ritornare l'attributo ActionTypes.

### 6.1.2 Front-end::ActionCreators

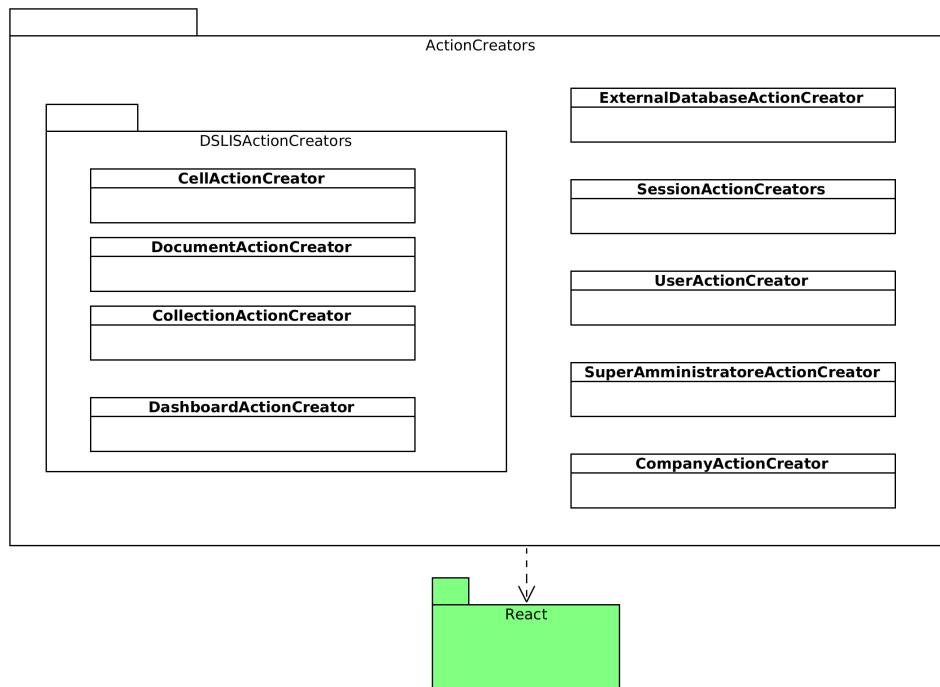


Figura 104: Diagramma delle classi del package ActionCreators

#### 6.1.2.1 Informazioni sul package

##### Descrizione

Package contenente le classi che si occupano di definire le liste di Action. Una Action è un oggetto che contiene i nuovi campi dati e il nome dell'azione da compiere. Questi oggetti rappresentano i messaggi inviati dalle diverse componenti dell'architettura del Front-end Flux.

##### Package contenuti

- DSLISActionCreators

##### Framework esterni

- React

### 6.1.2.2 Classi

#### UserActionCreator

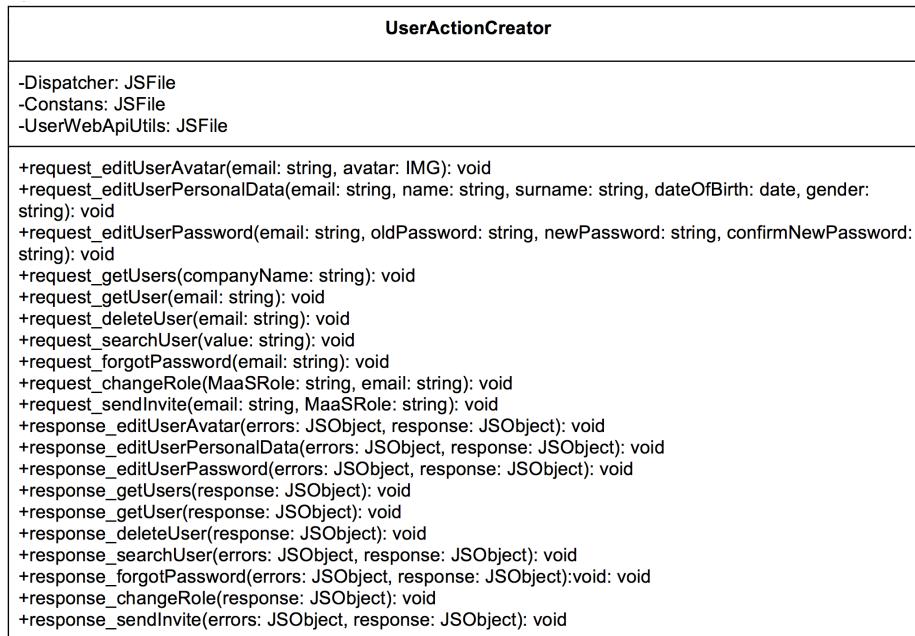


Figura 105: Diagramma della classe UserActionCreator

#### Descrizione

Classe che si occupa di creare la lista di Action riguardanti le interazioni degli utenti. Inoltre, essa si occupa, utilizzando il Dispatcher, di lanciare le Action verso la componente store. Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

#### Utilizzo

Viene utilizzata per creare e lanciare le Action riguardanti le azioni svolte dall'utente. Inizialmente, quando l'utente interagisce con la View per compiere delle operazioni riguardanti l'utente, l'UserActionCreator riceve il nome dell'azione compiuta e i relativi dati. Nel caso sia necessario, la classe utilizza i metodi forniti dall'UserWebAPIUtils per comunicare tali dati al server e ricevere altri dati da mostrare al client.

Successivamente, la classe impacchetta i dati e il nome dell'azione in un oggetto chiamato Action.

Infine, essa utilizza i metodi del Dispatcher per inoltrare l>Action alla relativa store, ovvero alla UserStore.

#### Relazioni con altre classi

- Front-end::WebAPIUtils::UserWebAPIUtils

- Front-end::Dispatcher

### Attributi

**-Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file dispatcher.js;

**-Constants : JSFile**

Questo attributo rappresenta un riferimento al file constants.js;

**-UserWebAPIUtils : JSFile**

Questo attributo rappresenta un riferimento al file UserWebAPIUtils.js.

### Metodi

**+request\_editUserAvatar(email: string, avatar: IMG): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe UserWebAPIUtils, fungendo quindi da tramite con la stessa;

**- email: string**

Questo parametro corrisponde all'email dell'utente a cui si vuole modificare l'avatar;

**- avatar: IMG**

Questo parametro corrisponde alla nuova immagine che fungerà da avatar dell'utente;

**+request\_editUserPersonalData(email: string, name: string, surname: string, dateOfBirth: date, gender: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe UserWebAPIUtils, fungendo quindi da tramite con la stessa;

**- email: string**

Questo parametro corrisponde all'email dell'utente a cui si vuole cambiare i dati anagrafici;

**- name: string**

Questo parametro corrisponde al nuovo nome dell'utente;

**- surname: string**

Questo parametro corrisponde al nuovo cognome dell'utente;

**- dateOfBirth: date**

Questo parametro corrisponde alla nuova data di nascita dell'utente;

**- gender: string**

Questo parametro corrisponde al nuovo sesso dell'utente;

**+request\_editUserPassword(email: string, oldPassword: string, newPassword: string, confirmPassword: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe UserWebAPIUtils, fungendo quindi da tramite con la stessa;

**- email: string**

Questo parametro corrisponde all'email dell'utente a cui si vuole cambiare la password;

```

- oldPassword: string
Questo parametro corrisponde alla vecchia password dell'utente;

- newPassword: string
Questo parametro corrisponde alla nuova password dell'utente;

- confirmNewPassword: string
Questo parametro corrisponde alla conferma della nuova password dell'utente;

+request_getUsers(companyName: string): void
Questo metodo si occupa di chiamare il metodo corrispondente della classe UserWebAPIUtils, fungendo quindi da tramite con la stessa;

- companyName: string
Questo parametro corrisponde al nome dell'azienda a cui l'utente appartiene;

+request_getUser(email: string): void
Questo metodo si occupa di chiamare il metodo corrispondente della classe UserWebAPIUtils, fungendo quindi da tramite con la stessa;

- email: string
Questo parametro corrisponde all'email dell'utente;

+request_deleteUser(email: string): void
Questo metodo si occupa di chiamare il metodo corrispondente della classe UserWebAPIUtils, fungendo quindi da tramite con la stessa;

- email: string
Questo parametro corrisponde all'email dell'utente da eliminare;

+request_searchUser(value: string): void
Questo metodo si occupa di chiamare il metodo corrispondente della classe UserWebAPIUtils, fungendo quindi da tramite con la stessa;

- value: string
Questo parametro corrisponde ad una generica stringa necessaria a cercare una serie di utenti;

+request_forgotPassword(email: string): void
Questo metodo si occupa di chiamare il metodo corrispondente della classe UserWebAPIUtils, fungendo quindi da tramite con la stessa;

- email: string
Questo parametro corrisponde all'email dell'utente che desidera recuperare la propria password;

+request_changeRole(MaaSRole: string, email: string): void
Questo metodo si occupa di chiamare il metodo corrispondente della classe UserWebAPIUtils, fungendo quindi da tramite con la stessa;

- MaaSRole: string
Questo parametro corrisponde al nuovo ruolo dell'utente;

- email: string
Questo parametro corrisponde all'email dell'utente interessato dal cambiamento di ruolo;

```

**+request\_sendInvite(email: string, MaaRole: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe UserWebAPIUtils, fungendo quindi da tramite con la stessa;

– **MaaRole: string**

Questo parametro corrisponde al ruolo del nuovo potenziale collaboratore dell'azienda;

– **email: string**

Questo parametro corrisponde all'email del nuovo potenziale collaboratore dell'azienda;

**+response\_editUserAvatar(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo

EDIT\_USER\_AVATAR\_RESPONSE e di inoltrarla alla relativa Store;

– **errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_editUserPersonalData(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo

EDIT\_USER\_PERSONAL\_DATA\_RESPONSE e di inoltrarla alla relativa Store;

– **errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_editUserPassword(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo

EDIT\_USER\_PASSWORD\_RESPONSE e di inoltrarla alla relativa Store;

– **errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_getUsers(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo

GET\_USERS\_RESPONSE e di inoltrarla alla relativa Store;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_getUser(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
GET\_USER\_RESPONSE e di inoltrarla alla relativa Store;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_deleteUser(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
DELETE\_USER\_RESPONSE e di inoltrarla alla relativa Store;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_searchUser(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
SEARCH\_USER\_RESPONSE e di inoltrarla alla relativa Store;

– **errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_forgotPassword(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
FORGOT\_PASSWORD\_RESPONSE e di inoltrarla alla relativa Store;

– **errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_changeRole(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
CHANGE\_ROLE\_RESPONSE e di inoltrarla alla relativa Store;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_sendInvite(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
SEND\_INVITE\_RESPONSE e di inoltrarla alla relativa Store;

– **errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

– **response: JSONObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server.

### SessionActionCreator

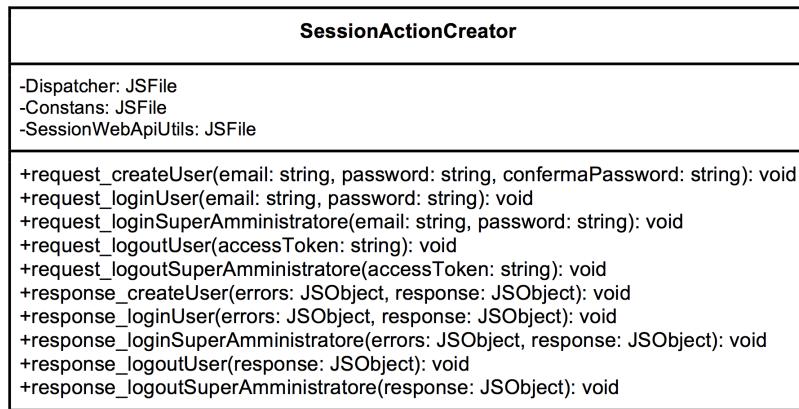


Figura 106: Diagramma della classe SessionActionCreator

### Descrizione

Classe che si occupa di creare la lista di Action riguardanti la sessione. Inoltre, essa si occupa, utilizzando il Dispatcher, di lanciare le Action verso la componente store. Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per creare e lanciare le Action riguardanti la sessione.

Inizialmente, quando l'utente interagisce con la View per compiere delle operazioni riguardanti la sessione (login, registrazione e logout), la classe SessionActionCreator riceve il nome dell'azione compiuta e i relativi dati. Nel caso sia necessario, la classe utilizza i metodi forniti da SessionWebAPIUtils per comunicare tali dati al server e ricevere altri dati da mostrare al client.

Successivamente, la classe impacchetta i dati e il nome dell'azione in un oggetto chiamato Action.

Infine, essa utilizza i metodi del Dispatcher per inoltrare l>Action alla relativa store, ovvero alla SessionStore.

### Relazioni con altre classi

- Front-end::WebAPIUtils::SessionWebAPIUtils
- Front-end::Dispatcher

## Attributi

### **-Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file dispatcher.js;

### **-Constants : JSFile**

Questo attributo rappresenta un riferimento al file constants.js;

### **-SessionWebAPIUtils : JSFile**

Questo attributo rappresenta un riferimento al file SessionWebAPIUtils.js.

## Metodi

### **+request\_createUser(email: string, password: string, confermaPassword: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe SessionWebAPIUtils, fungendo quindi da tramite con la stessa;

#### **- email: string**

Questo parametro corrisponde all'email utilizzata per registrare l'utente;

#### **- password: string**

Questo parametro corrisponde alla password utilizzata per registrare l'utente;

#### **- confermaPassword: string**

Questo parametro corrisponde alla conferma della password utilizzata per registrare l'utente;

### **+request\_loginUser(email: string, password: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe SessionWebAPIUtils, fungendo quindi da tramite con la stessa;

#### **- email: string**

Questo parametro corrisponde all'email dell'utente che vuole accedere al sistema;

#### **- password: string**

Questo parametro corrisponde alla password dell'utente che vuole accedere al sistema;

### **+request\_loginSuperAmministratore(email: string, password: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe SessionWebAPIUtils, fungendo quindi da tramite con la stessa;

#### **- email: string**

Questo parametro corrisponde all'email del Super Amministratore che vuole accedere al sistema;

#### **- password: string**

Questo parametro corrisponde alla password del Super Amministratore che vuole accedere al sistema;

### **+request\_logoutUser(accessToken: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe SessionWebAPIUtils, fungendo quindi da tramite con la stessa;

**- accessToken: string**

Questo parametro corrisponde all'accesstoken associato alla sessione di un utente;

**+request\_logoutSuperAmministratore(accessToken: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe SessionWebAPIUtils, fungendo quindi da tramite con la stessa;

**- accessToken: string**

Questo parametro corrisponde all'accesstoken associato alla sessione di un Super Amministratore;

**+response\_createUser(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo

CREATE\_USER\_RESPONSE e di inoltrarla alla relativa Store;

**- errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_loginUser(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo

LOGIN\_USER\_RESPONSE e di inoltrarla alla relativa Store;

**- errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_loginSuperAmministratore(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo

LOGIN\_SUPER\_AMMINISTRATORE\_RESPONSE e di inoltrarla alla relativa Store;

**- errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_logoutUser(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo

LOGOUT\_USER\_RESPONSE e di inoltrarla alla relativa Store;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

```
+response_logoutSuperAmministratore(response: JSObject): void
```

Questo metodo si occupa di creare una Action di tipo LOGOUT\_SUPER\_AMMINISTRATORE\_RESPONSE e di inoltrarla alla relativa Store;

- **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server.

### ExternalDatabaseActionCreator

ExternalDataBaseActionCreator
-Dispatcher: JSFile -Constans: JSFile -ExternalDatabaseWebApiUtils: JSFile
+request_createExternalDatabase(externalDatabaseName: string, configuration: JSON): void +request_deleteExternalDatabase(externalDatabaseID: number): void +request_getExternalDatabases(query: string): void +request_searchExternalDatabase(value: string, companyName: string): void +request_allowExternalDatabaseAccess(externalDatabaseID: number, email: string): void +request_denyExternalDatabaseAccess(externalDatabaseID: number, email: string): void +response_createExternalDatabase(errors: JSObject, response: JSObject): void +response_deleteExternalDatabase(response: JSObject): void +response_getExternalDatabases(response: JSObject): void +response_searchExternalDatabase(errors: JSObject, response: JSObject): void +response_allowExternalDatabaseAccess(response: JSObject): void +response_denyExternalDatabaseAccess(response: JSObject): void

Figura 107: Diagramma della classe ExternalDatabaseActionCreator

### Descrizione

Classe che si occupa di creare la lista di Action riguardanti l'insieme di azioni per gestire i database MongoDB esterni. Inoltre, essa si occupa, utilizzando il Dispatcher, di lanciare le Action verso la componente store.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per creare e lanciare le Action riguardanti l'insieme di azioni per gestire i database MongoDB esterni.

Inizialmente, quando l'utente interagisce con la View per compiere delle operazioni riguardanti i database MongoDB esterni, l'ExternalDatabaseActionCreator riceve il nome dell'azione compiuta e i relativi dati. Nel caso sia necessario, la classe utilizza i metodi forniti dall'ExternalDatabaseWebAPIUtils per comunicare tali dati al server e ricevere altri dati da mostrare al client.

Successivamente, la classe impacchetta i dati e il nome dell'azione in un oggetto chiamato Action.

Infine, essa utilizza i metodi del Dispatcher per inoltrare l>Action alla relativa store, ovvero alla ExternalDatabaseStore.

### Relazioni con altre classi

- Front-end::WebAPIUtils::ExternalDatabaseWebAPIUtils
- Front-end::Dispatcher

### Attributi

**-Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file dispatcher.js;

**-Constants : JSFile**

Questo attributo rappresenta un riferimento al file constants.js;

**-ExternalDatabaseWebAPIUtils : JSFile**

Questo attributo rappresenta un riferimento al file ExternalDatabaseWebAPIUtils.js.

### Metodi

**+request\_createExternalDatabase(externalDatabaseName: string, configuration: JSON): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe ExternalDatabaseWebAPIUtils, fungendo quindi da tramite con la stessa;

**- externalDatabaseName: string**

Questo parametro corrisponde al nome che si vuole dare al database esterno collegato;

**- configuration: JSON**

Questo parametro corrisponde ad un oggetto JSON contenente i dati di configurazione del database esterno collegato;

**+request\_deleteExternalDatabase(externalDatabaseID: number): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe ExternalDatabaseWebAPIUtils, fungendo quindi da tramite con la stessa;

**- externalDatabaseID: number**

Questo parametro corrisponde all'id del database esterno collegato che si vuole eliminare;

**+request\_getExternalDatabases(query: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe ExternalDatabaseWebAPIUtils, fungendo quindi da tramite con la stessa;

**- query: string**

Questo parametro corrisponde alla query utile a filtrare i database esterni appartenenti ad una sola azienda;

**+request\_searchExternalDatabase(value: string, companyName: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe ExternalDatabaseWebAPIUtils, fungendo quindi da tramite con la stessa;

**- value: string**

Questo parametro corrisponde ad una generica stringa necessaria a cercare una serie di database esterni;

**- `companyName: string`**

Questo parametro corrisponde ad un nome di un'azienda, utile a filtrare i database da cercare solo per una singola azienda;

**+`request_allowExternalDatabaseAccess(externalDatabaseID: number, email: string): void`**

Questo metodo si occupa di chiamare il metodo corrispondente della classe ExternalDatabaseWebAPIUtils, fungendo quindi da tramite con la stessa;

**- `externalDatabaseID: number`**

Questo parametro corrisponde all'id del database esterno a cui si vuole modificare i permessi di accesso;

**- `email: string`**

Questo parametro corrisponde all'email dell'utente a cui si vuole attribuire i permessi di accesso ad uno specifico database esterno;

**+`request_denyExternalDatabaseAccess(externalDatabaseID: number, email: string): void`**

Questo metodo si occupa di chiamare il metodo corrispondente della classe ExternalDatabaseWebAPIUtils, fungendo quindi da tramite con la stessa;

**- `externalDatabaseID: number`**

Questo parametro corrisponde all'id del database esterno a cui si vuole modificare i permessi di accesso;

**- `email: string`**

Questo parametro corrisponde all'email dell'utente a cui si vuole negare i permessi di accesso ad uno specifico database esterno;

**+`response_createExternalDatabase(errors: JSObject, response: JSObject): void`**

Questo metodo si occupa di creare una Action di tipo CREATE\_EXTERNAL\_DATABASE\_RESPONSE e di inoltrarla alla relativa Store;

**- `errors: JSObject`**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

**- `response: JSObject`**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+`response_deleteExternalDatabase(response: JSObject): void`**

Questo metodo si occupa di creare una Action di tipo DELETE\_EXTERNAL\_DATABASE\_RESPONSE e di inoltrarla alla relativa Store;

**- `response: JSObject`**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+`response_getExternalDatabases(response: JSObject): void`**

Questo metodo si occupa di creare una Action di tipo GET\_EXTERNAL\_DATABASE\_RESPONSE e di inoltrarla alla relativa Store;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_searchExternalDatabase(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo SEARCH\_EXTERNAL\_DATABASE\_RESPONSE e di inoltrarla alla relativa Store;

**- errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_allowExternalDatabaseAccess(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo ALLOW\_EXTERNAL\_DATABASE\_ACCESS\_RESPONSE e di inoltrarla alla relativa Store;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_denyExternalDatabaseAccess(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo DENY\_EXTERNAL\_DATABASE\_ACCESS\_RESPONSE e di inoltrarla alla relativa Store;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server.

### SuperAmministratoreActionCreator

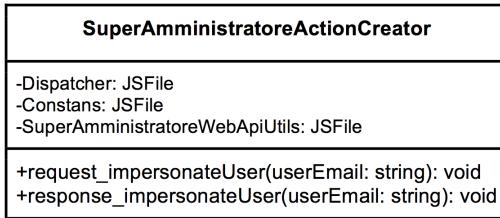


Figura 108: Diagramma della classe SuperAmministratoreActionCreator

### Descrizione

Classe che si occupa di creare la lista di Action riguardanti le interazioni del Super Amministratore. Inoltre, essa si occupa, utilizzando il Dispatcher, di lanciare le Action verso la componente store.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

## Utilizzo

Viene utilizzata per creare e lanciare le Action riguardanti le azioni svolte dal Super Amministratore.

Inizialmente, quando il Super Amministratore interagisce con la View per compiere delle operazioni, la classe SuperAmministratoreActionCreator riceve il nome dell'azione compiuta e i relativi dati. Nel caso sia necessario, la classe utilizza i metodi forniti da SuperAmministratoreWebAPIUtils per comunicare tali dati al server e ricevere altri dati da mostrare al client.

Successivamente, la classe impacchetta i dati e il nome dell'azione in un oggetto chiamato Action.

Infine, essa utilizza i metodi del Dispatcher per inoltrare l>Action alla relativa store, ovvero alla SuperAmministratoreStore.

## Relazioni con altre classi

- Front-end::WebAPIUtils::SuperAmministratoreWebAPIUtils
- Front-end::Dispatcher

## Attributi

**-Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file dispatcher.js;

**-Constants : JSFile**

Questo attributo rappresenta un riferimento al file constants.js;

**-SuperAmministratoreWebAPIUtils : JSFile**

Questo attributo rappresenta un riferimento al file SuperAmministratoreWebAPIUtils.js.

## Metodi

**+request\_impersonateUser(userEmail: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe SuperAmministratoreActionCreator, fungendo quindi da tramite con la stessa;

**- userEmail: string**

Questo parametro corrisponde all'email dell'utente che il Super Amministratore intende impersonificare;

**+response\_impersonateUser(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo IMPERSONATE\_USER\_RESPONSE e di inoltrarla alla relativa Store;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

## CompanyActionCreator

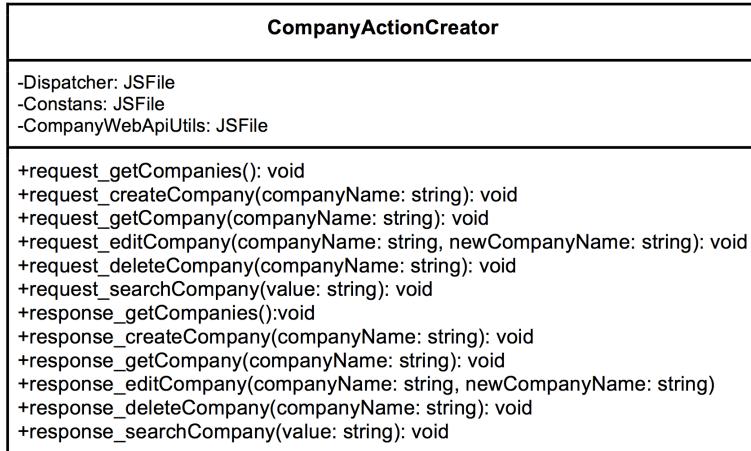


Figura 109: Diagramma della classe CompanyActionCreator

### Descrizione

Classe che si occupa di creare la lista di Action riguardanti le operazioni sui dati delle aziende. Inoltre, essa si occupa, utilizzando il Dispatcher, di lanciare le Action verso la componente store.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per creare e lanciare le Action riguardanti le operazioni per gestire i dati delle aziende.

Inizialmente, quando l'utente interagisce con la View per compiere delle operazioni riguardanti i dati delle aziende, la classe CompanyActionCreator riceve il nome dell'azione compiuta e i relativi dati. Nel caso sia necessario, la classe utilizza i metodi forniti da CompanyWebAPIUtils per comunicare tali dati al server e ricevere altri dati da mostrare al client.

Successivamente, la classe impacchetta i dati e il nome dell'azione in un oggetto chiamato Action.

Infine, essa utilizza i metodi del Dispatcher per inoltrare l>Action alla relativa store, ovvero alla CompanyStore.

### Relazioni con altre classi

- Front-end::WebAPIUtils::CompanyWebAPIUtils
- Front-end::Dispatcher

### Attributi

**-Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file dispatcher.js;

**-Constants : JSFile**

Questo attributo rappresenta un riferimento al file constants.js;

**-CompanyWebAPIUtils : JSFile**

Questo attributo rappresenta un riferimento al file CompanyWebAPIUtils.js.

## Metodi

**+request\_getCompanies(): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CompanyActionCreator, fungendo quindi da tramite con la stessa;

- Questo metodo non necessita di parametri;

**+request\_createCompany(companyName: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CompanyActionCreator, fungendo quindi da tramite con la stessa;

- **companyName: string**

Questo parametro corrisponde al nome della Company che deve essere creata;

**+request\_getCompany(companyName: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CompanyActionCreator, fungendo quindi da tramite con la stessa;

- **companyName: string**

Questo parametro corrisponde al nome della Company che deve essere trovata;

**+request\_editCompany(companyName: string, newCompanyName: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CompanyActionCreator, fungendo quindi da tramite con la stessa;

- **companyName: string**

Questo parametro corrisponde al nome della Company che deve essere trovata e modificata;

- **newCompanyName: string**

Questo parametro corrisponde al nuovo nome da attribuire alla Company;

**+request\_deleteCompany(companyName: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CompanyActionCreator, fungendo quindi da tramite con la stessa;

- **companyName: string**

Questo parametro corrisponde al nome della Company che deve essere eliminata;

**+request\_searchCompany(value: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CompanyActionCreator, fungendo quindi da tramite con la stessa;

- **value: string**

Questo parametro corrisponde al valore da cercare tra le Companies;

```
+response_getCompanies(response: JSObject):void
Questo metodo si occupa di creare una Action di tipo
GET_COMPANIES_RESPONSE e di inoltrarla alla relativa Store;

- response: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente il risultato della
richiesta al server;

+response_createCompany(errors: JSObject, response: JSObject)
Questo metodo si occupa di creare una Action di tipo
CREATE_COMPANY_RESPONSE e di inoltrarla alla relativa Store;

- errors: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti
dal server, in merito alla richiesta effettuata;

- response: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente il risultato della
richiesta al server;

+response_getCompany(errors: JSObject, response: JSObject)
Questo metodo si occupa di creare una Action di tipo
GET_COMPANY_RESPONSE e di inoltrarla alla relativa Store;

- errors: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti
dal server, in merito alla richiesta effettuata;

- response: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente il risultato della
richiesta al server;

+response_editCompany(errors: JSObject, response: JSObject)
Questo metodo si occupa di creare una Action di tipo
EDIT_COMPANY_RESPONSE e di inoltrarla alla relativa Store;

- errors: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti
dal server, in merito alla richiesta effettuata;

- response: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente il risultato della
richiesta al server;

+response_deleteCompany(errors: JSObject, response: JSObject)
Questo metodo si occupa di creare una Action di tipo
DELETE_COMPANY_RESPONSE e di inoltrarla alla relativa Store;

- errors: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti
dal server, in merito alla richiesta effettuata;

- response: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente il risultato della
richiesta al server;
```

### `+response_searchCompany(errors: JSObject, response: JSObject)`

Questo metodo si occupa di creare una Action di tipo SEARCH\_COMPANY\_RESPONSE e di inoltrarla alla relativa Store;

#### `- errors: JSObject`

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

#### `- response: JSObject`

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

### 6.1.3 Front-end::ActionCreators::DSLISActionCreators

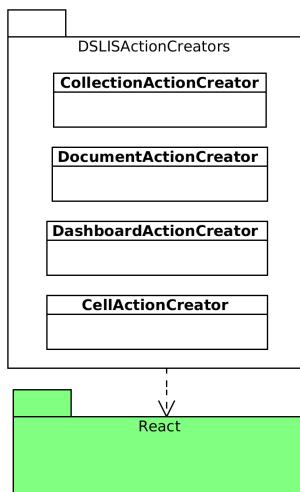


Figura 110: Diagramma delle classi del package DSLISActionCreators

#### 6.1.3.1 Informazioni sul package

##### Descrizione

Package contenente le classi che si occupano di definire le liste di Action riguardanti l'ambito dei DSLIS.

##### Framework esterni

- React

#### 6.1.3.2 Classi

##### CollectionActionCreator

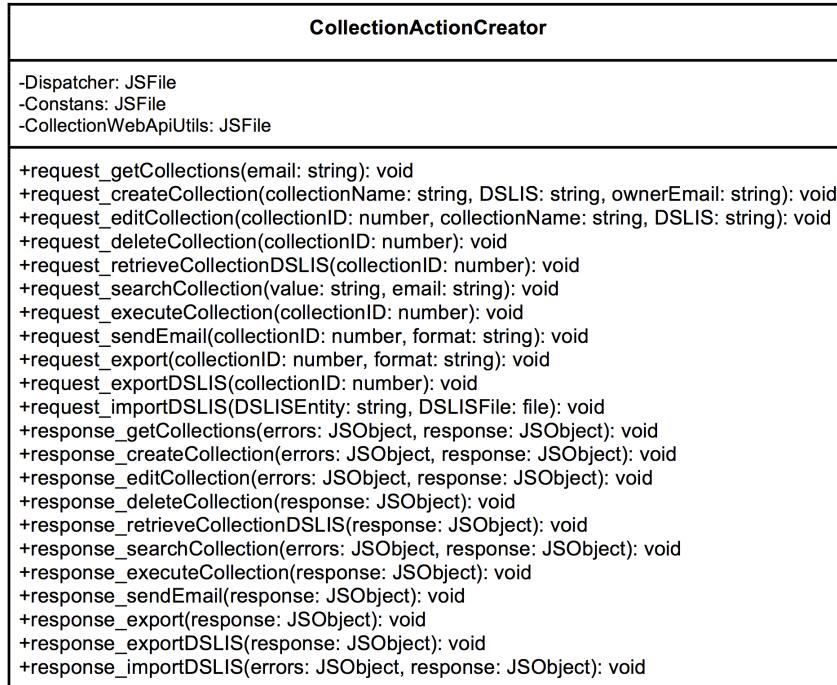


Figura 111: Diagramma della classe CollectionActionCreator

### Descrizione

Classe che si occupa di creare la lista di Action riguardanti le operazioni sulle Collection, create tramite DSLIS. Inoltre, essa si occupa, utilizzando il Dispatcher, di lanciare le Action verso la componente store.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per creare e lanciare le Action riguardanti le operazioni per gestire i dati delle Collection, create tramite DSLIS.

Inizialmente, quando l'utente interagisce con la View per compiere delle operazioni riguardanti le Collection, la classe CollectionActionCreator riceve il nome dell'azione compiuta e i relativi dati. Nel caso sia necessario, la classe utilizza i metodi forniti da CollectionWebAPIUtils per comunicare tali dati al server e ricevere altri dati da mostrare al client.

Successivamente, la classe impacchetta i dati e il nome dell'azione in un oggetto chiamato Action.

Infine, essa utilizza i metodi del Dispatcher per inoltrare l>Action alla relativa store, ovvero alla CollectionStore.

### Relazioni con altre classi

- Front-end::WebAPIUtils::DSLISWebAPIUtils::CollectionWebAPIUtils

- Front-end::Dispatcher

### Attributi

**-Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file dispatcher.js;

**-Constants : JSFile**

Questo attributo rappresenta un riferimento al file constants.js;

**-CollectionWebAPIUtils : JSFile**

Questo attributo rappresenta un riferimento al file CollectionWebAPIUtils.js.

### Metodi

**+request\_getCollections(email: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CollectionWebAPIUtils, fungendo quindi da tramite con la stessa;

**- email: string**

Questo parametro corrisponde all'email dell'utente del quale si vogliono cercare le collection, in particolare è lo stesso che provoca l'invocazione del metodo.

**+request\_createCollection(collectionName: string, DSLIS: string, ownerEmail: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CollectionWebAPIUtils, fungendo quindi da tramite con la stessa;

**- collectionName: string**

Questo parametro corrisponde al nome della nuova Collection che l'utente vuole creare;

**- DSLIS: string**

Questo parametro corrisponde al codice che compone la nuova Collection scritta dall'utente;

**- ownerEmail: string**

Questo parametro corrisponde all'email dell'autore della Collection.

**+request\_editCollection(collectionID: number, collectionName: string, DSLIS: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CollectionWebAPIUtils, fungendo quindi da tramite con la stessa;

**- collectionID: number**

Questo parametro corrisponde all'identificativo di della Collection oggetto della modifica;

**- collectionName: string**

Questo parametro corrisponde al nuovo nome della Collection oggetto della modifica;

**– `DSLIS: string`**

Questo parametro corrisponde al nuovo codice della Collection oggetto della modifica.

**+`request_deleteCollection(collectionID: number): void`**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CollectionWebAPIUtils, fungendo quindi da tramite con la stessa;

**– `collectionID: number`**

Questo parametro rappresenta l'identificativo della Collection che l'utente vuole eliminare;

**+`request_retrieveCollectionDSLIS(collectionID: number): void`**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CollectionWebAPIUtils, fungendo quindi da tramite con la stessa;

**– `collectionID: number`**

Questo parametro rappresenta l'identificativo della Collection della quale l'utente vuole il codice di cui è composta.

**+`request_searchCollection(value: string, email: string): void`**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CollectionWebAPIUtils, fungendo quindi da tramite con la stessa;

**– `value: string`**

Questo parametro corrisponde a una generica stringa necessaria a cercare una serie di Collection;

**– `email: string`**

Questo parametro corrisponde a all'indirizzo email dell'utente che desidera effettuare una ricerca.

**+`request_executeCollection(collectionID: number): void`**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CollectionWebAPIUtils, fungendo quindi da tramite con la stessa;

**– `collectionID: number`**

Questo parametro rappresenta l'identificativo della Collection che l'utente vuole eseguire.

**+`request_sendEmail(collectionID: number, format: string): void`**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CollectionWebAPIUtils, fungendo quindi da tramite con la stessa;

**– `collectionID: number`**

Questo parametro rappresenta l'identificativo della Collection oggetto dell'operazione;

**– `format: string`**

Questo parametro rappresenta il formato, a scelta tra CSV e JSON, con il quale l'utente desidera inviare la struttura dei dati della Collection selezionata.

**+`request_export(collectionID: number, format: string): void`**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CollectionWebAPIUtils, fungendo quindi da tramite con la stessa;

- **collectionID: number**  
Questo parametro rappresenta l'identificativo della Collection oggetto dell'operazione;
  - **format: string**  
Questo parametro rappresenta il formato, a scelta tra CSV e JSON, con il quale l'utente desidera esportare la struttura dei dati della Collection selezionata.
- +request\_exportDSLIS(collectionID: number): void**  
Questo metodo si occupa di chiamare il metodo corrispondente della classe CollectionWebAPIUtils, fungendo quindi da tramite con la stessa;
- **collectionID: number**  
Questo parametro rappresenta l'identificativo della Collection oggetto dell'operazione;
- +request\_importDSLIS(DSLISEntity: string, DSLISFile: file): void**  
Questo metodo si occupa di chiamare il metodo corrispondente della classe CollectionWebAPIUtils, fungendo quindi da tramite con la stessa;
- **DSLISEntity: string**  
Questo parametro rappresenta il tipo di DSLIS che l'utente desidera importare;
  - **DSLISFile: file**  
Questo parametro rappresenta il file che contiene il DSLIS che si desidera importare.
- +response\_getCollections(errors: JSObject, response: JSObject): void**  
Questo metodo si occupa di creare una Action di tipo  
GET\_COLLECTIONS\_RESPONSE e di inoltrarla alla relativa Store;
- **errors: JSObject**  
Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;
  - **response: JSObject**  
Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;
- +response\_createCollection(errors: JSObject, response: JSObject): void**  
Questo metodo si occupa di creare una Action di tipo  
CREATE\_COLLECTIONS\_RESPONSE e di inoltrarla alla relativa Store;
- **errors: JSObject**  
Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;
  - **response: JSObject**  
Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;
- +response\_editCollection(errors: JSObject, response: JSObject): void**  
Questo metodo si occupa di creare una Action di tipo  
EDIT\_COLLECTIONS\_RESPONSE e di inoltrarla alla relativa Store;

**- errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_deleteCollection(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
DELETE\_COLLECTIONS\_RESPONSE e di inoltrarla alla relativa Store;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_retrieveCollectionDSLIS(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
RETRIVE\_COLLECTIONS\_DSLIS\_RESPONSE e di inoltrarla alla relativa Store;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_searchCollection(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
SEARCH\_COLLECTIONS\_RESPONSE e di inoltrarla alla relativa Store;

**- errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_executeCollection(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
EXECUTE\_COLLECTIONS\_RESPONSE e di inoltrarla alla relativa Store;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_sendEmail(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
SEND\_EMAIL\_RESPONSE e di inoltrarla alla relativa Store;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_export(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
EXPORT\_RESPONSE e di inoltrarla alla relativa Store;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_exportDSLIS(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo EXPORT\_DSLIS\_RESPONSE e di inoltrarla alla relativa Store;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_importDSLIS(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo IMPORT\_DSLIS\_RESPONSE e di inoltrarla alla relativa Store;

**- errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

## DocumentActionCreator

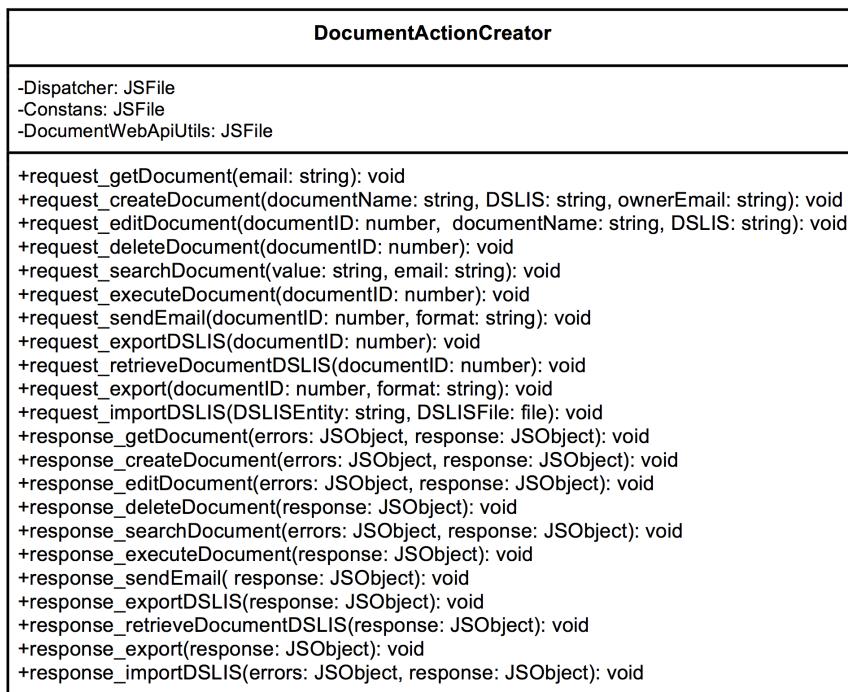


Figura 112: Diagramma della classe DocumentActionCreator

## Descrizione

Classe che si occupa di creare la lista di Action riguardanti le operazioni sui Document, creati tramite DSLIS. Inoltre, essa si occupa, utilizzando il Dispatcher, di lanciare le Action verso la componente store.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

## Utilizzo

Viene utilizzata per creare e lanciare le Action riguardanti le operazioni per gestire i dati dei Document, creati tramite DSLIS.

Inizialmente, quando l'utente interagisce con la View per compiere delle operazioni riguardanti i Document, la classe DocumentActionCreator riceve il nome dell'azione compiuta e i relativi dati. Nel caso sia necessario, la classe utilizza i metodi forniti da DocumentWebAPIUtils per comunicare tali dati al server e ricevere altri dati da mostrare al client.

Successivamente, la classe impacchetta i dati e il nome dell'azione in un oggetto chiamato Action.

Infine, essa utilizza i metodi del Dispatcher per inoltrare l>Action alla relativa store, ovvero alla DocumentStore.

## Relazioni con altre classi

- Front-end::WebAPIUtils::DSLISWebAPIUtils::DocumentWebAPIUtils
- Front-end::Dispatcher

## Attributi

### **-Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file dispatcher.js;

### **-Constants : JSFile**

Questo attributo rappresenta un riferimento al file constants.js;

### **-DocumentWebAPIUtils : JSFile**

Questo attributo rappresenta un riferimento al file DocumentWebAPIUtils.js.

## Metodi

### **+request\_getDocument(email: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe DocumentWebAPIUtils, fungendo quindi da tramite con la stessa;

#### **- email: string**

Questo parametro corrisponde all'email dell'utente del quale si vogliono cercare i documenti, in particolare è lo stesso utente che provoca l'invocazione del metodo.

### **+request\_createDocument(documentName: string, DSLIS: string, ownerEmail: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe DocumentWebAPIUtils, fungendo quindi da tramite con la stessa;

– **documentName: string**

Questo parametro corrisponde al nome del nuovo Document che l'utente vuole creare;

– **DSLIS: string**

Questo parametro corrisponde al codice che compone il nuovo Document scritto dall'utente;

– **ownerEmail: string**

Questo parametro corrisponde all'email dell'autore del Document.

```
+request_editDocument(documentID: number, documentName: string, DSLIS: string): void
```

Questo metodo si occupa di chiamare il metodo corrispondente della classe DocumentWebAPIUtils, fungendo quindi da tramite con la stessa;

– **documentID: number**

Questo parametro corrisponde all'identificativo di un Document oggetto della modifica;

– **documentName: string**

Questo parametro corrisponde al nuovo nome del Document oggetto della modifica;

– **DSLIS: string**

Questo parametro corrisponde al nuovo codice del Document oggetto della modifica.

```
+request_deleteDocument(documentID: number): void
```

Questo metodo si occupa di chiamare il metodo corrispondente della classe DocumentWebAPIUtils, fungendo quindi da tramite con la stessa;

– **documentID: number**

Questo parametro rappresenta l'identificativo del Document che l'utente vuole eliminare;

```
+request_searchDocument(value: string, email: string): void
```

Questo metodo si occupa di chiamare il metodo corrispondente della classe DocumentWebAPIUtils, fungendo quindi da tramite con la stessa;

– **value: string**

Questo parametro corrisponde a una generica stringa necessaria a cercare una serie di Document;

– **email: string**

Questo parametro corrisponde a all'indirizzo email dell'utente che desidera effettuare una ricerca.

```
+request_executeDocument(DocumentID: number): void
```

Questo metodo si occupa di chiamare il metodo corrispondente della classe DocumentWebAPIUtils, fungendo quindi da tramite con la stessa;

– **documentID: number**

Questo parametro rappresenta l'identificativo del Document che l'utente vuole eseguire.

**+request\_sendEmail(documentID: number, format: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe DocumentWebAPIUtils, fungendo quindi da tramite con la stessa;

– **DocumentID: number**

Questo parametro rappresenta l'identificativo del Document oggetto dell'operazione;

– **format: string**

Questo parametro rappresenta il formato, a scelta tra CSV e JSON, con il quale l'utente desidera inviare la struttura dei dati del Document selezionato.

**+request\_exportDSLIS(DocumentID: number): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe DocumentWebAPIUtils, fungendo quindi da tramite con la stessa;

– **documentID: number**

Questo parametro rappresenta l'identificativo della Document oggetto dell'operazione;

**+request\_retrieveDocumentDSLIS(documentID: number): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe DocumentWebAPIUtils, fungendo quindi da tramite con la stessa;

– **documentID: number**

Questo parametro rappresenta l'identificativo del Document del quale l'utente vuole il codice di cui è composto.

**+request\_export(documentID: number, format: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe DocumentWebAPIUtils, fungendo quindi da tramite con la stessa;

– **documentID: number**

Questo parametro rappresenta l'identificativo del Document oggetto dell'operazione;

– **format: string**

Questo parametro rappresenta il formato, a scelta tra CSV e JSON, con il quale l'utente desidera esportare la struttura dei dati del Document selezionato.

**+importDSLIS(DSLISEntity: string, DSLISFile: file): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe DocumentWebAPIUtils, fungendo quindi da tramite con la stessa;

– **DSLISEntity: string**

Questo parametro rappresenta il tipo di DSLIS che l'utente desidera importare;

– **DSLISFile: file**

Questo parametro rappresenta il file che contiene il DSLIS che si desidera importare.

**+response\_getDocument(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo

GET\_DOCUMENT\_RESPONSE e di inoltrarla alla relativa Store;

– **errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

**– response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_createDocument(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo CREATE\_DOCUMENT\_RESPONSE e di inoltrarla alla relativa Store;

**– errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

**– response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_editDocument(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo EDIT\_DOCUMENT\_RESPONSE e di inoltrarla alla relativa Store;

**– errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

**– response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_deleteDocument(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo DELETE\_DOCUMENT\_RESPONSE e di inoltrarla alla relativa Store;

**– response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_searchDocument(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo SEARCH\_DOCUMENT\_RESPONSE e di inoltrarla alla relativa Store;

**– errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

**– response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_executeDocument(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo EXECUTE\_DOCUMENT\_RESPONSE e di inoltrarla alla relativa Store;

**– response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_sendEmail( response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
SEND\_EMAIL\_RESPONSE e di inoltrarla alla relativa Store;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della  
richiesta al server;

**+response\_exportDSLIS(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
EXPORT\_DSLIS\_RESPONSE e di inoltrarla alla relativa Store;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della  
richiesta al server;

**+response\_retrieveDocumentDSLIS(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
RETRIEVE\_DOCUMENT\_DSLIS\_RESPONSE e di inoltrarla alla relativa Store;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della  
richiesta al server;

**+response\_export(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
EXPORT\_RESPONSE e di inoltrarla alla relativa Store;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della  
richiesta al server;

**+response\_importDSLIS(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
IMPORT\_DSLIS\_RESPONSE e di inoltrarla alla relativa Store;

**- errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti  
dal server, in merito alla richiesta effettuata;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della  
richiesta al server;

## DashboardActionCreator

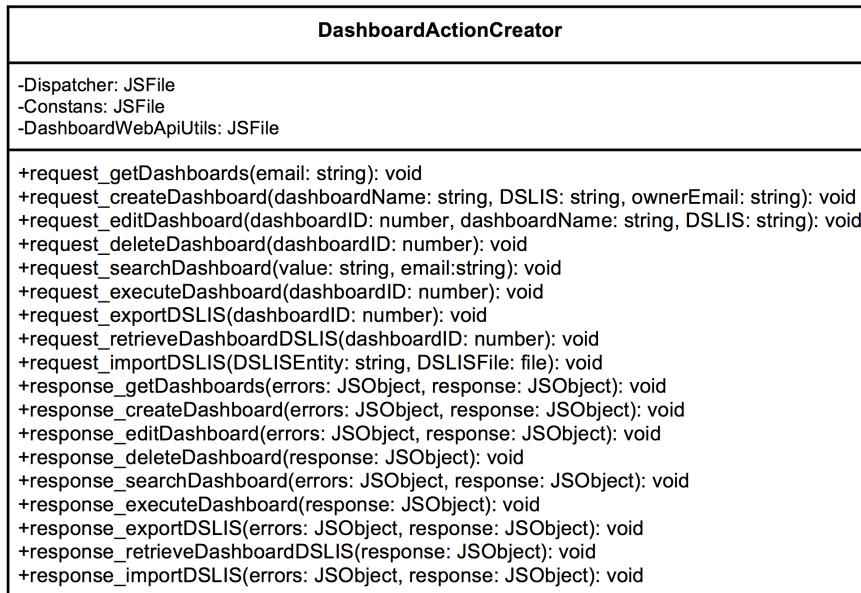


Figura 113: Diagramma della classe DashboardActionCreator

### Descrizione

Classe che si occupa di creare la lista di Action riguardanti le operazioni sulle Dashboard, create tramite DSLIS. Inoltre, essa si occupa, utilizzando il Dispatcher, di lanciare le Action verso la componente store.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per creare e lanciare le Action riguardanti le operazioni per gestire i dati delle Dashboard, create tramite DSLIS.

Inizialmente, quando l'utente interagisce con la View per compiere delle operazioni riguardanti le Dashboard, la classe DashboardActionCreator riceve il nome dell'azione compiuta e i relativi dati. Nel caso sia necessario, la classe utilizza i metodi forniti da DashboardWebAPIUtils per comunicare tali dati al server e ricevere altri dati da mostrare al client.

Successivamente, la classe impacchetta i dati e il nome dell'azione in un oggetto chiamato Action.

Infine, essa utilizza i metodi del Dispatcher per inoltrare l>Action alla relativa store, ovvero alla DashboardStore.

### Relazioni con altre classi

- Front-end::WebAPIUtils::DSLISWebAPIUtils::DashboardWebAPIUtils
- Front-end::Dispatcher

## Attributi

### **-Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file dispatcher.js;

### **-Constants : JSFile**

Questo attributo rappresenta un riferimento al file constants.js;

### **-DashboardWebAPIUtils : JSFile**

Questo attributo rappresenta un riferimento al file DashboardWebAPIUtils.js.

## Metodi

### **+request\_getDashboard(email: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe DashboardWebAPIUtils, fungendo quindi da tramite con la stessa;

#### **- email: string**

Questo parametro corrisponde all'email dell'utente del quale si vogliono cercare le Dashboard, in particolare è lo stesso utente che provoca l'invocazione del metodo.

### **+request\_createDashboard(dashboardName: string, DSLIS: string, ownerEmail: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe DashboardWebAPIUtils, fungendo quindi da tramite con la stessa;

#### **- dashboardName: string**

Questo parametro corrisponde al nome della nuova Dashboard che l'utente vuole creare;

#### **- DSLIS: string**

Questo parametro corrisponde al codice che compone la nuova Dashboard scritta dall'utente;

#### **- ownerEmail: string**

Questo parametro corrisponde all'email dell'autore della Dashboard.

### **+request\_editDashboard(dashboardID: number, dashboardName: string, DSLIS: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe DashboardWebAPIUtils, fungendo quindi da tramite con la stessa;

#### **- dashboardID: number**

Questo parametro corrisponde all'identificativo di una Dashboard oggetto della modifica;

#### **- dashboardName: string**

Questo parametro corrisponde al nuovo nome della Dashboard oggetto della modifica;

#### **- DSLIS: string**

Questo parametro corrisponde al nuovo codice della Dashboard oggetto della modifica.

**+request\_deleteDashboard(dashboardID: number): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe DashboardWebAPIUtils, fungendo quindi da tramite con la stessa;

**- dashboardID: number**

Questo parametro rappresenta l'identificativo della Dashboard che l'utente vuole eliminare;

**+request\_searchDashboard(value: string, email: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe DashboardWebAPIUtils, fungendo quindi da tramite con la stessa;

**- value: string**

Questo parametro corrisponde a una generica stringa necessaria a cercare una serie di Dashboard;

**- email: string**

Questo parametro corrisponde a all'indirizzo email dell'utente che desidera effettuare una ricerca.

**+request\_executeDashboard(dashboardID: number): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe DashboardWebAPIUtils, fungendo quindi da tramite con la stessa;

**- dashboardID: number**

Questo parametro rappresenta l'identificativo della Dashboard che l'utente vuole eseguire.

**+request\_exportDSLIS(dashboardID: number): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe DashboardWebAPIUtils, fungendo quindi da tramite con la stessa;

**- dashboardID: number**

Questo parametro rappresenta l'identificativo della Dashboard oggetto dell'operazione;

**+request\_retrieveDashboardDSLIS(dashboardID: number): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe DashboardWebAPIUtils, fungendo quindi da tramite con la stessa;

**- dashboardID: number**

Questo parametro rappresenta l'identificativo della Dashboard del quale l'utente vuole il codice di cui è composta.

**+request\_importDSLIS(DSLISEntity: string, DSLISFile: file): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe DashboardWebAPIUtils, fungendo quindi da tramite con la stessa;

**- DSLISEntity: string**

Questo parametro rappresenta il tipo di DSLIS che l'utente desidera importare;

**- DSLISFile: file**

Questo parametro rappresenta il file che contiene il DSLIS che si desidera importare.

```
+response_getDashboards(errors: JSObject, response: JSObject): void
Questo metodo si occupa di creare una Action di tipo
GET_DASHBOARDS_RESPONSE e di inoltrarla alla relativa Store;

- errors: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti
dal server, in merito alla richiesta effettuata;

- response: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente il risultato della
richiesta al server;
```

```
+response_createDashboard(errors: JSObject, response: JSObject): void
Questo metodo si occupa di creare una Action di tipo
CREATE_DASHBOARD_RESPONSE e di inoltrarla alla relativa Store;

- errors: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti
dal server, in merito alla richiesta effettuata;

- response: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente il risultato della
richiesta al server;
```

```
+response_editDashboard(errors: JSObject, response: JSObject): void
Questo metodo si occupa di creare una Action di tipo
EDIT_DASHBOARD_RESPONSE e di inoltrarla alla relativa Store;

- errors: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti
dal server, in merito alla richiesta effettuata;

- response: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente il risultato della
richiesta al server;
```

```
+response_deleteDashboard(response: JSObject): void
Questo metodo si occupa di creare una Action di tipo
DELETE_DASHBOARD_RESPONSE e di inoltrarla alla relativa Store;

- response: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente il risultato della
richiesta al server;
```

```
+response_searchDashboard(errors: JSObject, response: JSObject): void
Questo metodo si occupa di creare una Action di tipo
SEARCH_DASHBOARD_RESPONSE e di inoltrarla alla relativa Store;

- errors: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti
dal server, in merito alla richiesta effettuata;

- response: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente il risultato della
richiesta al server;
```

**+response\_executeDashboard(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo EXECUTE\_DASHBOARD\_RESPONSE e di inoltrarla alla relativa Store;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_exportDSLIS(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo EXPORT\_DSLIS\_RESPONSE e di inoltrarla alla relativa Store;

– **errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_retrieveDashboardDSLIS(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo RETRIEVE\_DASHBOARD\_RESPONSE e di inoltrarla alla relativa Store;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_importDSLIS(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo IMPORT\_DSLIS\_RESPONSE e di inoltrarla alla relativa Store;

– **errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

## CellActionCreator

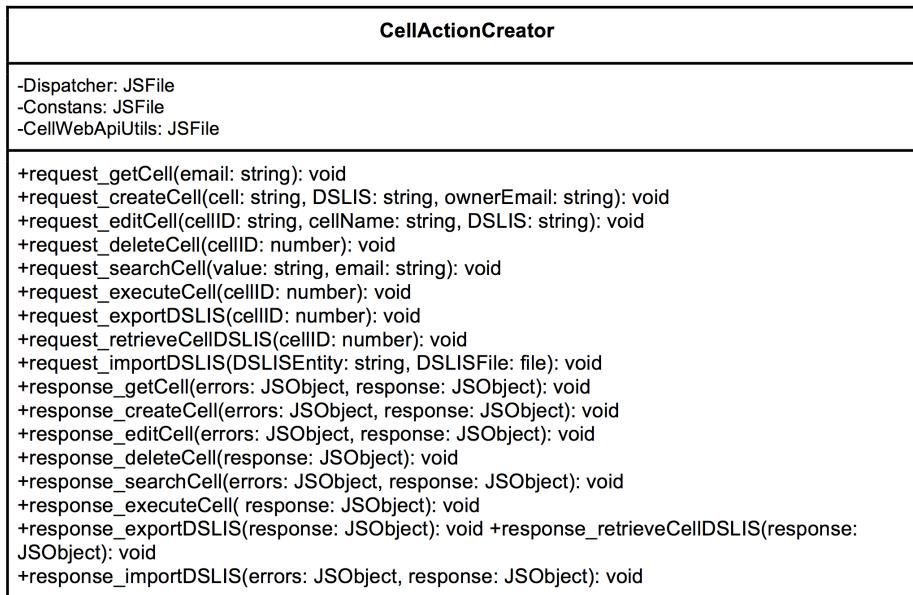


Figura 114: Diagramma della classe CellActionCreator

### Descrizione

Classe che si occupa di creare la lista di Action riguardanti le operazioni sulle Cell, create tramite DSLIS. Inoltre, essa si occupa, utilizzando il Dispatcher, di lanciare le Action verso la componente store.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per creare e lanciare le Action riguardanti le operazioni per gestire i dati delle Cell, create tramite DSLIS.

Inizialmente, quando l'utente interagisce con la View per compiere delle operazioni riguardanti le Cell, la classe CellActionCreator riceve il nome dell'azione compiuta e i relativi dati. Nel caso sia necessario, la classe utilizza i metodi forniti da CellWebAPIUtils per comunicare tali dati al server e ricevere altri dati da mostrare al client.

Successivamente, la classe impacchetta i dati e il nome dell'azione in un oggetto chiamato Action.

Infine, essa utilizza i metodi del Dispatcher per inoltrare l>Action alla relativa store, ovvero alla CellStore.

### Relazioni con altre classi

- Front-end::WebAPIUtils::DSLISWebAPIUtils::CellWebAPIUtils
- Front-end::Dispatcher

## Attributi

### **-Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file dispatcher.js;

### **-Constants : JSFile**

Questo attributo rappresenta un riferimento al file constants.js;

### **-CellWebAPIUtils : JSFile**

Questo attributo rappresenta un riferimento al file CellWebAPIUtils.js.

## Metodi

### **+request\_getCell(email: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CellWebAPIUtils, fungendo quindi da tramite con la stessa;

#### **- email: string**

Questo parametro corrisponde all'email dell'utente del quale si vogliono cercare le Cell, in particolare è lo stesso utente che provoca l'invocazione del metodo.

### **+request\_createCell(cellName: string, DSLIS: string, ownerEmail: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CellWebAPIUtils, fungendo quindi da tramite con la stessa;

#### **- cellName: string**

Questo parametro corrisponde al nome della nuova Cell che l'utente vuole creare;

#### **- DSLIS: string**

Questo parametro corrisponde al codice che compone la nuova Cell scritta dall'utente;

#### **- ownerEmail: string**

Questo parametro corrisponde all'email dell'autore della Cell.

### **+request\_editCell(cellID: number, cellName: string, DSLIS: string): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CellWebAPIUtils, fungendo quindi da tramite con la stessa;

#### **- cellID: number**

Questo parametro corrisponde all'identificativo di una Cell oggetto della modifica;

#### **- cellName: string**

Questo parametro corrisponde al nuovo nome della Cell oggetto della modifica;

#### **- DSLIS: string**

Questo parametro corrisponde al nuovo codice della Cell oggetto della modifica.

### **+request\_deleteCell(cellID: number): void**

Questo metodo si occupa di chiamare il metodo corrispondente della classe CellWebAPIUtils, fungendo quindi da tramite con la stessa;

#### **- cellID: number**

Questo parametro rappresenta l'identificativo della Cell che l'utente vuole eliminare;

```
+request_searchCell(value: string, email: string): void
Questo metodo si occupa di chiamare il metodo corrispondente della classe
CellWebAPIUtils, fungendo quindi da tramite con la stessa;

- value: string
Questo parametro corrisponde a una generica stringa necessaria a cercare una serie
di Cell;

- email: string
Questo parametro corrisponde a all'indirizzo email dell'utente che desidera effettuare
una ricerca.

+request_executeCell(cellID: number): void
Questo metodo si occupa di chiamare il metodo corrispondente della classe
CellWebAPIUtils, fungendo quindi da tramite con la stessa;

- cellID: number
Questo parametro rappresenta l'identificativo della Cell che l'utente vuole eseguire.

+request_exportDSLIS(cellID: number): void
Questo metodo si occupa di chiamare il metodo corrispondente della classe
CellWebAPIUtils, fungendo quindi da tramite con la stessa;

- cellID: number
Questo parametro rappresenta l'identificativo della Cell oggetto dell'operazione;

+request_retrieveCellDSLIS(cellID: number): void
Questo metodo si occupa di chiamare il metodo corrispondente della classe
CellWebAPIUtils, fungendo quindi da tramite con la stessa;

- cellID: number
Questo parametro rappresenta l'identificativo della Cell del quale l'utente vuole il
codice di cui è composto.

+request_importDSLIS(DSLISEntity: string, DSLISFile: file): void
Questo metodo si occupa di chiamare il metodo corrispondente della classe
CellWebAPIUtils, fungendo quindi da tramite con la stessa;

- DSLISEntity: string
Questo parametro rappresenta il tipo di DSLIS che l'utente desidera importare;

- DSLISFile: file
Questo parametro rappresenta il file che contiene il DSLIS che si desidera
importare.

+response_getCell(errors: JSObject, response: JSObject): void
Questo metodo si occupa di creare una Action di tipo
GET_CELL_RESPONSE e di inoltrarla alla relativa Store;

- errors: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti
dal server, in merito alla richiesta effettuata;

- response: JSObject
Questo parametro corrisponde all'oggetto Javascript contenente il risultato della
richiesta al server;
```

**+response\_createCell(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
CREATE\_CELL\_RESPONSE e di inoltrarla alla relativa Store;

– **errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_editCell(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
EDIT\_CELL\_RESPONSE e di inoltrarla alla relativa Store;

– **errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_deleteCell(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
DELETE\_CELL\_RESPONSE e di inoltrarla alla relativa Store;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_searchCell(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
SEARCH\_CELL\_RESPONSE e di inoltrarla alla relativa Store;

– **errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_executeCell(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
EXECUTE\_CELL\_RESPONSE e di inoltrarla alla relativa Store;

– **response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_exportDSLIS(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo  
EXPORT\_DSLIS\_RESPONSE e di inoltrarla alla relativa Store;

**- errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_retrieveCellDSLIS(response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo RETRIEVE\_CELL\_RESPONSE e di inoltrarla alla relativa Store;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

**+response\_importDSLIS(errors: JSObject, response: JSObject): void**

Questo metodo si occupa di creare una Action di tipo IMPORT\_DSLIS\_RESPONSE e di inoltrarla alla relativa Store;

**- errors: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente gli errori provenienti dal server, in merito alla richiesta effettuata;

**- response: JSObject**

Questo parametro corrisponde all'oggetto Javascript contenente il risultato della richiesta al server;

#### 6.1.4 Front-end::WebAPIUtils

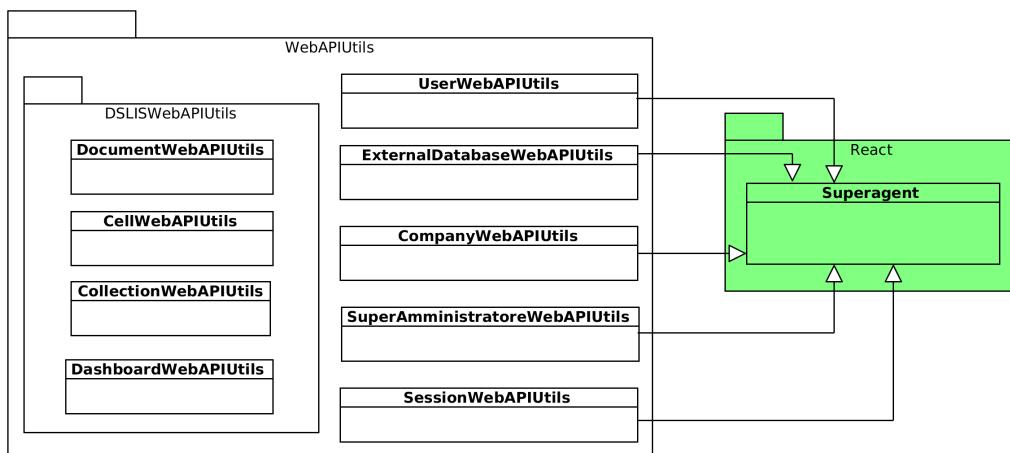


Figura 115: Diagramma delle classi del package WEBAPIUtils

##### 6.1.4.1 Informazioni sul package

###### Descrizione

Package contenente le classi che si occupano di fornire dei metodi per interfacciarsi con le API fornite dal server da parte del client. Fondamentalmente, sono classi che forniscono metodi che permettono di inviare delle richieste e ricevere delle risposte, in modo da permettere al client di comunicare con il server.

### Package contenuti

- DSLISWebAPIUtils

#### 6.1.4.2 Classi

##### UserWebAPIUtils

<b>UserWebAPIUtils</b>
<pre>-UserActionCreator : JSFile -Constants : JSFile</pre>
<pre>+editUserAvatar(email: string, avatar: IMG): void +editUserPersonalData(email: string, name: string, surname: string, dateOfBirth: date, gender: string): void +editUserPassword(email: string, oldPassword: string, newPassword: string, confirmPassword: string): void +getUsers(companyName: string): void +getUser(email: string): void +deleteUser(email: string): void +searchUser(value: string): void +forgotPassword(email: string): void +changeRole(MaaSRole: string, email: string): void +sendInvite(email: string, MaaSRole: string): void -getErrors(res: JSON): string</pre>

Figura 116: Diagramma della classe UserWebAPIUtils

### Descrizione

Classe che fornisce i metodi per inviare richieste e ricevere risposte nell'ambito delle operazioni svolte dall'utente, in modo da permettere al client di comunicare con il server.  
Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per fornire dei metodi utili ad inviare le richieste HTTP verso le API fornite dal server e riceverne le risposte da parte di esso, nell'ambito delle operazioni svolte da parte dell'utente. Le risposte ricevute saranno inoltrate verso l'UserActionCreator, che andrà a creare una Action corrispondente.

### Classi ereditate

- React::Superagent

## Relazioni con altri classi

- Front-end::Constants

## Attributi

**-Constants : JSFile**

Questo attributo rappresenta un riferimento al file constants.js;

**-UserActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js.

## Metodi

**-getErrors(res: JSON): string**

Questo metodo si occupa di gestire gli errori provenienti dal server. Viene utilizzato dalle funzioni della classe di appartenenza che gestiscono le varie richieste;

**- res: JSON**

Questo parametro corrisponde all'oggetto JSON, rappresentante la risposta del server ad una richiesta;

**+editUserAvatar(email: string, avatar: IMG): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente i dati della form di modifica dell'avatar, sostituendolo con quello appena caricato dall'utente. Successivamente, riceve una risposta dal back-end contenente un errore, oppure una conferma di successo dell'operazione;

**- email: string**

Questo parametro corrisponde all'email dell'utente a cui si vuole cambiare l'avatar;

**- avatar: IMG**

Questo parametro corrisponde alla nuova immagine che fungerà da avatar dell'utente;

**+editUserPersonalData(email: string, name: string, surname: string, dateOfBirth: date, gender: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente i dati della form di modifica dei dati anagrafici, sostituendoli con quelli appena inseriti dall'utente. Successivamente, riceve una risposta dal back-end contenente un errore, oppure una conferma di successo dell'operazione;

**- email: string**

Questo parametro corrisponde all'email dell'utente a cui si vuole cambiare i dati anagrafici;

**- name: string**

Questo parametro corrisponde al nuovo nome dell'utente;

**- surname: string**

Questo parametro corrisponde al nuovo cognome dell'utente;

**- dateOfBirth: date**

Questo parametro corrisponde alla nuova data di nascita dell'utente;

**- gender: string**

Questo parametro corrisponde al nuovo sesso dell'utente;

**+editUserPassword(email: string, oldPassword: string, newPassword: string, confirmPassword: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente i dati della form di modifica della password, sostituendola con quella appena caricata dall'utente. Successivamente, riceve una risposta dal back-end contenente un errore, oppure una conferma di successo dell'operazione;

**- email: string**

Questo parametro corrisponde all'email dell'utente a cui si vuole cambiare la password;

**- oldPassword: string**

Questo parametro corrisponde alla vecchia password dell'utente;

**- newPassword: string**

Questo parametro corrisponde alla nuova password dell'utente;

**- confirmPassword: string**

Questo parametro corrisponde alla conferma della nuova password dell'utente;

**+getUsers(companyName: string) : void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente il nome di un'azienda della quale mostrare gli utenti che ne fanno parte. Successivamente, riceve una risposta dal back-end contenente i dati degli stessi;

**- companyName: string**

Questo parametro corrisponde al nome dell'azienda a cui l'utente appartiene;

**+getUser(email: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'email di un utente, utile a mostrare i suoi dati. Successivamente, riceve una risposta dal back-end contenente i dati dello stesso;

**- email: string**

Questo parametro corrisponde all'email dell'utente;

**+deleteUser(email: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'email di un utente, che si vuole eliminare dal sistema. Successivamente, riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

**- email: string**

Questo parametro corrisponde all'email dell'utente da eliminare;

**+searchUser(value: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente una stringa generica, necessaria alla ricerca di una lista di utenti. Successivamente, riceve una risposta dal back-end contenente un errore, oppure la lista composta dagli utenti che corrispondono alla ricerca effettuata;

**- value: string**

Questo parametro corrisponde ad una generica stringa necessaria a cercare una serie di utenti;

**+forgotPassword(email: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'email di un utente, di cui si vuole recuperare la password. Successivamente, riceve una risposta dal back-end contenente un errore, oppure una conferma di successo dell'operazione;

**- email: string**

Questo parametro corrisponde all'email dell'utente che desidera recuperare la propria password;

**+changeRole(MaaSRole: string, email: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'email di un utente ed il nuovo ruolo, che egli andrà a ricoprire all'interno del sistema MaaS. Successivamente, riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

**- MaaSRole: string**

Questo parametro corrisponde al nuovo ruolo dell'utente;

**- email: string**

Questo parametro corrisponde all'email dell'utente interessato dal cambiamento di ruolo;

**+sendInvite(email: string, MaaSRole: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente un'email ed un ruolo, utile a invitare un potenziale nuovo collaboratore dell'azienda. Successivamente, riceve una risposta dal back-end contenente un errore, oppure una conferma di successo dell'operazione;

**- MaaSRole: string**

Questo parametro corrisponde al ruolo del nuovo potenziale collaboratore dell'azienda;

**- email: string**

Questo parametro corrisponde all'email del nuovo potenziale collaboratore dell'azienda.

## SessionWebAPIUtils

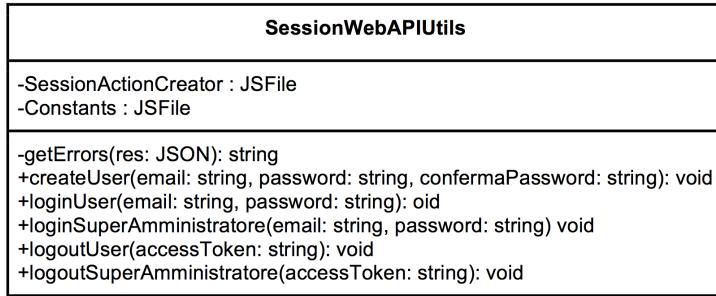


Figura 117: Diagramma della classe SessionWebAPIUtils

### Descrizione

Classe che fornisce i metodi per inviare richieste e ricevere risposte nell'ambito delle operazioni svolte per la sessione, in modo da permettere al client di comunicare con il server. In particolare, operazioni quali login, registrazione e logout. Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per fornire dei metodi utili ad inviare le richieste HTTP verso le API fornite dal server e riceverne le risposte da parte di esso, nell'ambito delle operazioni svolte per la sessione. Le risposte ricevute saranno inoltrate verso la classe SessionActionCreator, che andrà a creare una Action corrispondente.

### Classi ereditate

- React::Superagent

### Relazioni con altri classi

- Front-end::Constants

### Attributi

**-Constants : JSFile**

Questo attributo rappresenta un riferimento al file constants.js;

**-SessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreator.js.

### Metodi

**-getErrors(res: JSON): string**

Questo metodo si occupa di gestire gli errori provenienti dal server. Viene utilizzato dalle funzioni della classe di appartenenza che gestiscono le varie richieste;

**- `res: JSON`**

Questo parametro corrisponde all'oggetto JSON, rappresentante la risposta del server ad una richiesta;

**+`createUser(email: string, password: string, confermaPassword: string): void`**

Questo metodo si occupa di inviare una richiesta HTTP, contenente un'email, una password e la conferma della password, utili alla registrazione di un utente. Successivamente, riceve una risposta dal back-end contenente un errore, oppure la conferma di successo dell'operazione;

**- `email: string`**

Questo parametro corrisponde all'email utilizzata per registrare l'utente;

**- `password: string`**

Questo parametro corrisponde alla password utilizzata per registrare l'utente;

**- `confermaPassword: string`**

Questo parametro corrisponde alla conferma della password utilizzata per registrare l'utente;

**+`loginUser(email: string, password: string): void`**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'email e la password, utili per effettuare l'accesso di un utente al sistema. Successivamente, riceve una risposta dal back-end contenente un errore, oppure una conferma di successo dell'operazione;

**- `email: string`**

Questo parametro corrisponde all'email dell'utente che vuole accedere al sistema;

**- `password: string`**

Questo parametro corrisponde alla password dell'utente che vuole accedere al sistema;

**+`loginSuperAmministratore(email: string, password: string): void`**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'email e la password, utili per effettuare l'accesso di un Super Amministratore al sistema. Successivamente, riceve una risposta dal back-end contenente un errore, oppure una conferma di successo dell'operazione;

**- `email: string`**

Questo parametro corrisponde all'email del Super Amministratore che vuole accedere al sistema;

**- `password: string`**

Questo parametro corrisponde alla password del Super Amministratore che vuole accedere al sistema;

**+`logoutUser(accessToken: string): void`**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'accesstoken associato ad un utente, in modo da rimuovere la sessione ad esso associato e così eseguire il logout. Successivamente, riceve una risposta dal back-end contenente la conferma di successo dell'operazione;

**- `accessToken: string`**

Questo parametro corrisponde all'accesstoken associato alla sessione di un utente;

### `+logoutSuperAmministratore(accessToken: string): void`

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'accesstoken associato ad un Super Amministratore, in modo da rimuovere la sessione ad esso associato e così eseguire il logout. Successivamente, riceve una risposta dal back-end contenente la conferma di successo dell'operazione;

- `accessToken: string`

Questo parametro corrisponde all'accesstoken associato alla sessione di un Super Amministratore.

## ExternalDatabaseWebAPIUtils

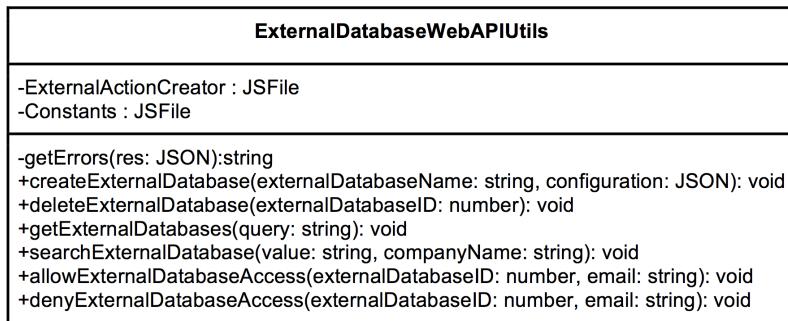


Figura 118: Diagramma della classe ExternalDatabaseWebAPIUtils

### Descrizione

Classe che fornisce i metodi per inviare richieste e ricevere risposte nell'ambito delle operazioni svolte per gestire i database MongoDB esterni, in modo da permettere al client di comunicare con il server.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per fornire dei metodi utili ad inviare le richieste HTTP verso le API fornite dal server e riceverne le risposte da parte di esso, nell'ambito delle operazioni svolte per gestire i database MongoDB esterni. Le risposte ricevute saranno inoltrate verso l'ExternalDatabaseActionCreator, che andrà a creare una Action corrispondente.

### Classi ereditate

- React:::Superagent

### Relazioni con altri classi

- Front-end::Constants

## Attributi

### **-Constants : JSFile**

Questo attributo rappresenta un riferimento al file constants.js;

### **-ExternalDatabaseActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file ExternalDatabaseActionCreator.js.

## Metodi

### **-getErrors(res: JSON): string**

Questo metodo si occupa di gestire gli errori provenienti dal server. Viene utilizzato dalle funzioni della classe di appartenenza che gestiscono le varie richieste;

#### **- res: JSON**

Questo parametro corrisponde all'oggetto JSON, rappresentante la risposta del server ad una richiesta;

### **+createExternalDatabase(externalDatabaseName: string, configuration: JSON): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente i dati della form di connessione ad un database esterno, utili a creare una nuova istanza di un database esterno. Successivamente, riceve una risposta dal back-end contenente un errore, oppure una conferma di successo dell'operazione;

#### **- externalDatabaseName: string**

Questo parametro corrisponde al nome che si vuole dare al database esterno collegato;

#### **- configuration: JSON**

Questo parametro corrisponde ad un oggetto JSON contenente i dati di configurazione del database esterno collegato;

### **+deleteExternalDatabase(externalDatabaseID: number): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'id del database esterno, che si vuole eliminare. Successivamente, riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

#### **- externalDatabaseID: number**

Questo parametro corrisponde all'id del database esterno collegato che si vuole eliminare;

### **+getExternalDatabases(query: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente la query che permette di filtrare i database esterni di una sola azienda e di restituirne i dati. Successivamente, riceve una risposta dal back-end contenente i dati da mostrare;

#### **- query: string**

Questo parametro corrisponde alla query utile a filtrare i database esterni appartenenti ad una sola azienda;

### **+searchExternalDatabase(value: string, companyName: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente una stringa generica ed il nome di un'azienda, necessari alla ricerca di una lista di database esterni.

Successivamente, riceve una risposta dal back-end contenente un errore, oppure la lista composta dai database esterni che corrispondono alla ricerca effettuata;

– **value: string**

Questo parametro corrisponde ad una generica stringa necessaria a cercare una serie di database esterni;

– **companyName: string**

Questo parametro corrisponde ad un nome di un'azienda, utile a filtrare i database da cercare solo per una singola azienda;

**+allowExternalDatabaseAccess(externalDatabaseID: number, email: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'id di un database esterno e la email di un utente, necessari all'attribuzione dei permessi di accesso di un database esterno ad un utente. Successivamente, riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

– **externalDatabaseID: number**

Questo parametro corrisponde all'id del database esterno a cui si vuole modificare i permessi di accesso;

– **email: string**

Questo parametro corrisponde all'email dell'utente a cui si vuole attribuire i permessi di accesso ad uno specifico database esterno;

**+denyExternalDatabaseAccess(externalDatabaseID: number, email: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'id di un database esterno e la email di un utente, necessari alla rimozione dei permessi di accesso di un database esterno ad un utente. Successivamente, riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

– **externalDatabaseID: number**

Questo parametro corrisponde all'id del database esterno a cui si vuole modificare i permessi di accesso;

– **email: string**

Questo parametro corrisponde all'email dell'utente a cui si vuole negare i permessi di accesso ad uno specifico database esterno.

## CompanyWebAPIUtils

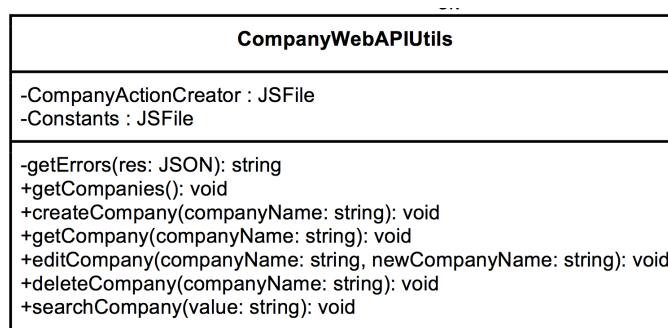


Figura 119: Diagramma della classe CompanyWebAPIUtils

### Descrizione

Classe che fornisce i metodi per inviare richieste e ricevere risposte nell'ambito delle operazioni svolte per gestire i dati delle aziende, in modo da permettere al client di comunicare con il server.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per fornire dei metodi utili ad inviare le richieste HTTP verso le API fornite dal server e riceverne le risposte da parte di esso, nell'ambito delle operazioni svolte per gestire i dati delle aziende. Le risposte ricevute saranno inoltrate verso la classe CompanyActionCreator, che andrà a creare una Action corrispondente.

### Classi ereditate

- React::Superagent

### Relazioni con altri classi

- Front-end::Constants

### Attributi

**-Constants : JSFile**

Questo attributo rappresenta un riferimento al file constants.js;

**-CompanyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js.

### Metodi

**-getErrors(res: JSON): string**

Questo metodo si occupa di gestire gli errori provenienti dal server. Viene utilizzato dalle funzioni della classe di appartenenza che gestiscono le varie richieste;

**- res: JSON**

Questo parametro corrisponde all'oggetto JSON, rappresentante la risposta del server ad una richiesta;

**+getCompanies(): void**

Questo metodo si occupa di inviare una richiesta HTTP utile a richiedere i dati delle aziende iscritte al sistema. Successivamente, riceve una risposta dal back-end contenente i dati da mostrare;

**+createCompany(companyName: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente il nome dell'azienda, che si vuole creare. Successivamente, riceve una risposta dal back-end contenente un errore, oppure una conferma di successo dell'operazione;

**- companyName: string**

Questo parametro corrisponde al nome dell'azienda che si vuole creare;

**+getCompany(companyName: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente il nome di un'azienda, in modo da mostrarne i dati. Successivamente, riceve una risposta dal back-end contenente i dati dell'azienda da mostrare;

**- companyName: string**

Questo parametro corrisponde al nome dell'azienda di cui si necessitano i dati per mostrarla;

**+editCompany(companyName: string, newCompanyName: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente il nome dell'azienda e il nuovo nome da assegnarle, in modo da modificare i dati di essa. Successivamente, riceve una risposta dal back-end contenente un errore, oppure una conferma di successo dell'operazione;

**- companyName: string**

Questo parametro corrisponde al nome dell'azienda a cui si vuole modificare i dati;

**- newCompanyName: string**

Questo parametro corrisponde al nuovo nome da assegnare ad un'azienda;

**+deleteCompany(companyName: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente il nome di un'azienda, necessario all'eliminazione di essa. Successivamente, riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

**- companyName: string**

Questo parametro corrisponde al nome dell'azienda che si intende eliminare;

**+searchCompany(value: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente una stringa generica, necessaria alla ricerca di una lista di aziende. Successivamente, riceve una risposta dal back-end contenente un errore, oppure la lista composta dalle aziende che corrispondono alla ricerca effettuata;

**- value: string**

Questo parametro corrisponde ad una stringa generica, utile alla ricerca di un'azienda.

## SuperAmministratoreWebAPIUtils

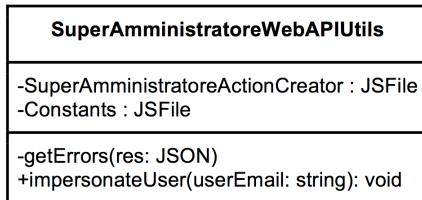


Figura 120: Diagramma della classe SuperAmministratoreWebAPIUtils

### Descrizione

Classe che fornisce i metodi per inviare richieste e ricevere risposte nell'ambito delle operazioni svolte dal Super Amministratore, in modo da permettere al client di comunicare con il server. Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per fornire dei metodi utili ad inviare le richieste HTTP verso le API fornite dal server e riceverne le risposte da parte di esso, nell'ambito delle operazioni svolte dal Super Amministratore. Le risposte ricevute saranno inoltrate verso la classe SuperAmministratoreActionCreator, che andrà a creare una Action corrispondente.

### Classi ereditate

- React::Superagent

### Relazioni con altri classi

- Front-end::Constants

### Attributi

**-Constants : JSFile**

Questo attributo rappresenta un riferimento al file constants.js;

**-SuperAmministratoreActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SuperAmministratoreActionCreator.js.

### Metodi

**-getErrors(res: JSON): string**

Questo metodo si occupa di gestire gli errori provenienti dal server. Viene utilizzato dalle funzioni della classe di appartenenza che gestiscono le varie richieste;

**- res: JSON**

Questo parametro corrisponde all'oggetto JSON, rappresentante la risposta del server ad una richiesta;

```
+impersonateUser(userEmail: string): void
```

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'email di un utente, utile ad impersonare un'utente. Successivamente, riceve una risposta dal back-end contenente la conferma di successo dell'operazione;

- `userEmail: string`

Questo parametro corrisponde all'email di un utente, il quale si vuole impersonare.

### 6.1.5 Front-end::WebAPIUtils::DSLISWebAPIUtils

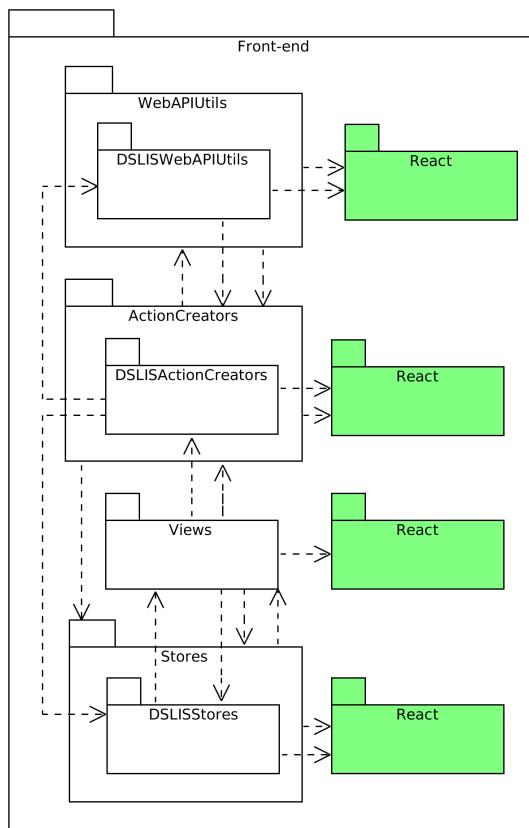


Figura 121: Diagramma delle classi del package DSLISWebAPIUtils

#### 6.1.5.1 Informazioni sul package

##### Descrizione

Package contenente le classi che si occupano di fornire dei metodi per interfacciarsi con le API fornite dal server da parte del client, nell'ambito delle operazioni riguardanti i DSLIS.

### 6.1.5.2 Classi

#### CollectionWebAPIUtils

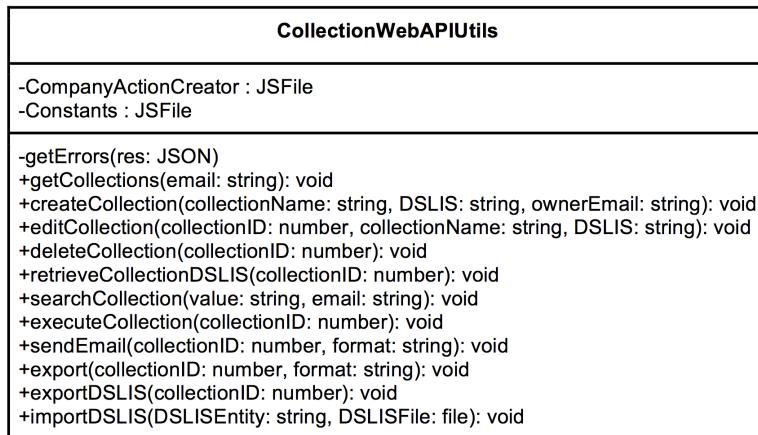


Figura 122: Diagramma della classe CollectionWebAPIUtils

#### Descrizione

Classe che fornisce i metodi per inviare richieste e ricevere risposte nell'ambito delle operazioni sulle Collection create mediante i DSLIS, in modo da permettere al client di comunicare con il server.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

#### Utilizzo

Viene utilizzata per fornire dei metodi utili ad inviare le richieste HTTP verso le API fornite dal server e riceverne le risposte da parte di esso, nell'ambito delle operazioni sulle Collection. Le risposte ricevute saranno inoltrate verso la classe CollectionActionCreator, che andrà a creare una Action corrispondente.

#### Classi ereditate

- React::Superagent

#### Relazioni con altri classi

- Front-end::Constants

#### Attributi

**-Constants : JSfile**

Questo attributo rappresenta un riferimento al file constants.js;

**-CollectionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CollectionActionCreator.js.

## Metodi

**-getErrors(res: JSON): string**

Questo metodo si occupa di gestire gli errori provenienti dal server. Viene utilizzato dalle funzioni della classe di appartenenza che gestiscono le varie richieste;

**- res: JSON**

Questo parametro corrisponde all'oggetto JSON, rappresentante la risposta del server ad una richiesta;

**+getCollections(email: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'indirizzo email di un utente, necessario alla ricerca della lista delle Collection in suo possesso. Successivamente, riceve una risposta dal back-end contenente un errore, oppure la lista composta dalle Collection in possesso dello stesso;

**- email: string**

Questo parametro corrisponde all'email dell'utente del quale si vogliono cercare le Collection, in particolare è lo stesso che provoca l'invocazione del metodo.

**+createCollection(collectionName: string, DSLIS: string, ownerEmail: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente il nome della nuova Collection, il codice che la compone (DSLIS) e l'email dell'autore, con lo scopo di creare una nuova Collection. Successivamente, riceve una risposta dal back-end contenente un errore o la conferma di successo dell'operazione;

**- collectionName: string**

Questo parametro corrisponde al nome della nuova Collection che l'utente vuole creare;

**- DSLIS: string**

Questo parametro corrisponde al codice che compone la nuova Collection scritta dall'utente;

**- ownerEmail: string**

Questo parametro corrisponde all'email dell'autore della Collection.

**+editCollection(collectionID: number, collectionName: string, DSLIS: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'identificativo di una Collection, il suo nome e il nuovo codice che la compone, utili a modificare una Collection già esistente. Successivamente, riceve una risposta dal back-end contenente un errore o la conferma di successo dell'operazione;

**- collectionID: number**

Questo parametro corrisponde all'identificativo di della Collection oggetto della modifica;

– **collectionName: string**

Questo parametro corrisponde al nuovo nome della Collection oggetto della modifica;

– **DSLIS: string**

Questo parametro corrisponde al nuovo codice della Collection oggetto della modifica.

**+deleteCollection(collectionID: number): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'identificativo di una Collection. Successivamente, riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

– **collectionID: number**

Questo parametro rappresenta l'identificativo della Collection che l'utente vuole eliminare;

**+retrieveCollectionDSLIS(collectionID: number): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'identificativo di una Collection, utile a ritornare il codice della stessa. Successivamente, riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

– **collectionID: number**

Questo parametro rappresenta l'identificativo della Collection della quale l'utente vuole il codice di cui è composta.

**+searchCollection(value: string, email: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente una stringa generica necessaria alla ricerca di una lista di Collection e l'email relativa all'utente che esegue la ricerca. Successivamente riceve una risposta dal back-end contenente un errore, oppure la lista composta dalle Collection che corrispondono alla ricerca effettuata;

– **value: string**

Questo parametro corrisponde a una generica stringa necessaria a cercare una serie di Collection;

– **email: string**

Questo parametro corrisponde a all'indirizzo email dell'utente che desidera effettuare una ricerca.

**+executeCollection(collectionID: number): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente un identificativo di una Collection, necessario ad eseguirla. Successivamente riceve una risposta dal back-end contenente il risultato dell'operazione in formato JSON;

– **collectionID: number**

Questo parametro rappresenta l'identificativo della Collection che l'utente vuole eseguire.

**+sendEmail(collectionID: number, format: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente un identificativo di una Collection e una stringa che rappresenta il formato, necessari a inviare per mezzo di email, la struttura dei dati della Collection selezionata nel formato indicato.

Successivamente riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

– **collectionID: number**

Questo parametro rappresenta l'identificativo della Collection oggetto dell'operazione;

– **format: string**

Questo parametro rappresenta il formato, a scelta tra CSV e JSON, con il quale l'utente desidera inviare la struttura dei dati della Collection selezionata.

**+export(collectionID: number, format: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente un identificativo di una Collection e una stringa che rappresenta il formato di esportazione, necessari a esportare la struttura dei dati della Collection selezionata nel formato voluto. Successivamente riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

– **collectionID: number**

Questo parametro rappresenta l'identificativo della Collection oggetto dell'operazione;

– **format: string**

Questo parametro rappresenta il formato, a scelta tra CSV e JSON, con il quale l'utente desidera esportare la struttura dei dati della Collection selezionata.

**+exportDSLIS(collectionID: number): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente un identificativo di una Collection, necessario a esportare il DSLIS corrispondente alla Collection indicata. Successivamente riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

– **collectionID: number**

Questo parametro rappresenta l'identificativo della Collection oggetto dell'operazione;

**+importDSLIS(DSLISEntity: string, DSLISfile: file): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente una stringa che rappresenta il tipo di DSLIS che si vuole importare e il file che contiene il DSLIS da importare. Successivamente, riceve una risposta dal back-end contenente un errore o la conferma di successo dell'operazione;

– **DSLISEntity: string**

Questo parametro rappresenta il tipo di DSLIS che l'utente desidera importare;

– **DSLISfile: file**

Questo parametro rappresenta il file che contiene il DSLIS che si desidera importare.

## DocumentWebAPIUtils

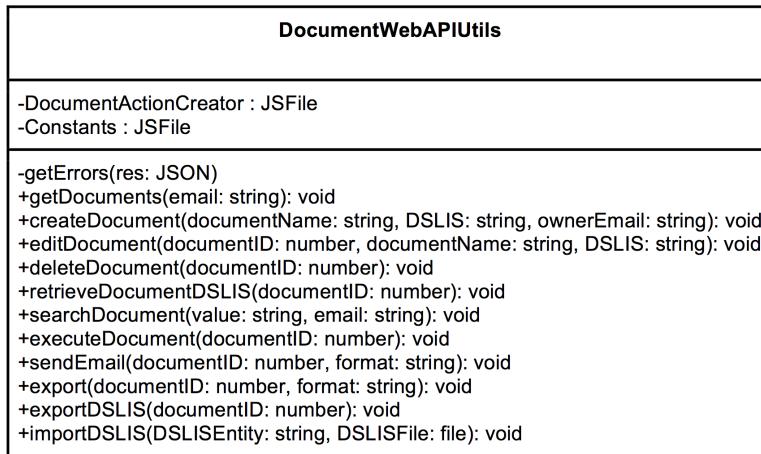


Figura 123: Diagramma della classe DocumentWebAPIUtils

### Descrizione

Classe che fornisce i metodi per inviare richieste e ricevere risposte nell'ambito delle operazioni sui Document creati mediante i DSLIS, in modo da permettere al client di comunicare con il server.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per fornire dei metodi utili ad inviare le richieste HTTP verso le API fornite dal server e riceverne le risposte da parte di esso, nell'ambito delle operazioni sui Document. Le risposte ricevute saranno inoltrate verso la classe DocumentActionCreator, che andrà a creare una Action corrispondente.

### Classi ereditate

- React::Superagent

### Relazioni con altri classi

- Front-end::Constants

### Attributi

**-Constants : JSfile**

Questo attributo rappresenta un riferimento al file constants.js;

**-DocumentActionCreator : JSfile**

Questo attributo rappresenta un riferimento al file DocumentActionCreator.js.

## Metodi

**-getErrors(res: JSON): string**

Questo metodo si occupa di gestire gli errori provenienti dal server. Viene utilizzato dalle funzioni della classe di appartenenza che gestiscono le varie richieste;

**- res: JSON**

Questo parametro corrisponde all'oggetto JSON, rappresentante la risposta del server ad una richiesta;

**+getDocument(email: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'indirizzo email di un utente, necessario alla ricerca della lista dei documenti in suo possesso. Successivamente, riceve una risposta dal back-end contenente un errore, oppure la lista composta dei documenti in possesso dello stesso;

**- email: string**

Questo parametro corrisponde all'email dell'utente del quale si vogliono cercare i documenti, in particolare è lo stesso utente che provoca l'invocazione del metodo.

**+createDocument(documentName: string, DSLIS: string, ownerEmail: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente il nome del nuovo Document, il codice che lo compone (DSLIS) e l'email dell'autore, con lo scopo di creare un nuovo Document. Successivamente, riceve una risposta dal back-end contenente un errore o la conferma di successo dell'operazione;

**- documentName: string**

Questo parametro corrisponde al nome del nuovo Document che l'utente vuole creare;

**- DSLIS: string**

Questo parametro corrisponde al codice che compone il nuovo Document scritto dall'utente;

**- ownerEmail: string**

Questo parametro corrisponde all'email dell'autore del Document.

**+editDocument(documentID: number, documentName: string, DSLIS: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'identificativo di un Document, il suo nome e il nuovo codice che lo compone, utili a modificare un Document già esistente. Successivamente, riceve una risposta dal back-end contenente un errore o la conferma di successo dell'operazione;

**- documentID: number**

Questo parametro corrisponde all'identificativo di un Document oggetto della modifica;

**- documentName: string**

Questo parametro corrisponde al nuovo nome del Document oggetto della modifica;

**- DSLIS: string**

Questo parametro corrisponde al nuovo codice del Document oggetto della modifica.

**+deleteDocument(documentID: number): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'identificativo di un document. Successivamente, riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

**- documentID: number**

Questo parametro rappresenta l'identificativo del Document che l'utente vuole eliminare;

**+searchDocument(value: string, email: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente una stringa generica necessaria alla ricerca di una lista di Document e l'email relativa all'utente che esegue la ricerca. Successivamente riceve una risposta dal back-end contenente un errore, oppure la lista composta dai Document che corrispondono alla ricerca effettuata;

**- value: string**

Questo parametro corrisponde a una generica stringa necessaria a cercare una serie di Document;

**- email: string**

Questo parametro corrisponde a all'indirizzo email dell'utente che desidera effettuare una ricerca.

**+executeDocument(documentID: number): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente un identificativo di un Document, necessario ad eseguirlo. Successivamente riceve una risposta dal back-end contenente il risultato dell'operazione in formato JSON;

**- documentID: number**

Questo parametro rappresenta l'identificativo del Document che l'utente vuole eseguire.

**+sendEmail(documentID: number, format: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente un identificativo di un Document e una stringa che rappresenta il formato, necessari a inviare, per mezzo di email, la struttura dei dati del Document selezionata nel formato indicato. Successivamente riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

**- DocumentID: number**

Questo parametro rappresenta l'identificativo del Document oggetto dell'operazione;

**- format: string**

Questo parametro rappresenta il formato, a scelta tra CSV e JSON, con il quale l'utente desidera inviare la struttura dei dati del Document selezionato.

**+exportDSLIS(DocumentID: number): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente un identificativo di un Document, necessario a esportare il DSLIS corrispondente alla Document indicata. Successivamente riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

**- `documentID: number`**

Questo parametro rappresenta l'identificativo della Document oggetto dell'operazione;

**+`retrieveDocumentDSLIS(documentID: number): void`**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'identificativo di un Document, utile a ritornare il codice dello stesso. Successivamente, riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

**- `documentID: number`**

Questo parametro rappresenta l'identificativo del Document del quale l'utente vuole il codice di cui è composto.

**+`export(documentID: number, format: string): void`**

Questo metodo si occupa di inviare una richiesta HTTP, contenente un identificativo di un Document e una stringa che rappresenta il formato di esportazione, necessari a esportare la struttura dei dati del Document selezionato nel formato voluto. Successivamente riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

**- `documentID: number`**

Questo parametro rappresenta l'identificativo del Document oggetto dell'operazione;

**- `format: string`**

Questo parametro rappresenta il formato, a scelta tra CSV e JSON, con il quale l'utente desidera esportare la struttura dei dati del Document selezionato.

**+`importDSLIS(DSLISEntity: string, DSLISFile: file): void`**

Questo metodo si occupa di inviare una richiesta HTTP, contenente una stringa che rappresenta il tipo di DSLIS che si vuole importare e il file che contiene il DSLIS da importare. Successivamente riceve una risposta dal back-end contenente un errore o una conferma di successo dell'operazione;

**- `DSLISEntity: string`**

Questo parametro rappresenta il tipo di DSLIS che l'utente desidera importare;

**- `DSLISFile: file`**

Questo parametro rappresenta il file che contiene il DSLIS che si desidera importare.

## DashboardWebAPIUtils

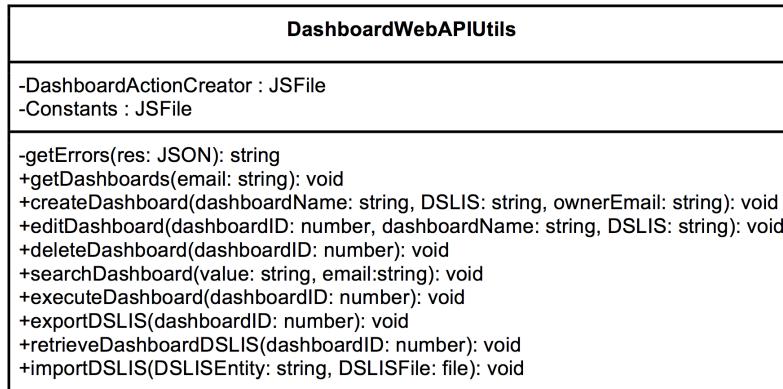


Figura 124: Diagramma della classe DashboardWebAPIUtils

### Descrizione

Classe che fornisce i metodi per inviare richieste e ricevere risposte nell'ambito delle operazioni sulle Dashboard create mediante i DSLIS, in modo da permettere al client di comunicare con il server.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per fornire dei metodi utili ad inviare le richieste HTTP verso le API fornite dal server e riceverne le risposte da parte di esso, nell'ambito delle operazioni sulle Dashboard. Le risposte ricevute saranno inoltrate verso la classe DashboardActionCreator, che andrà a creare una Action corrispondente.

### Classi ereditate

- React::Superagent

### Relazioni con altri classi

- Front-end::Constants

### Attributi

**-Constants : JSFile**

Questo attributo rappresenta un riferimento al file constants.js;

**-DashboardActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file DashboardActionCreator.js.

## Metodi

**-getErrors(res: JSON): string**

Questo metodo si occupa di gestire gli errori provenienti dal server. Viene utilizzato dalle funzioni della classe di appartenenza che gestiscono le varie richieste;

**- res: JSON**

Questo parametro corrisponde all'oggetto JSON, rappresentante la risposta del server ad una richiesta;

**+getDashboard(email: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'indirizzo email di un utente, necessario alla ricerca della lista delle Dashboard in suo possesso. Successivamente, riceve una risposta dal back-end contenente un errore, oppure la lista composta dalle Dashboard in possesso dello stesso;

**- email: string**

Questo parametro corrisponde all'email dell'utente del quale si vogliono cercare le Dashboard, in particolare è lo stesso utente che provoca l'invocazione del metodo.

**+createDashboard(dashboardName: string, DSLIS: string, ownerEmail: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente il nome della nuova Dashboard, il codice che la compone (DSLIS) e l'email dell'autore, con lo scopo di creare una nuova Dashboard. Successivamente, riceve una risposta dal back-end contenente un errore o la conferma di successo dell'operazione;

**- dashboardName: string**

Questo parametro corrisponde al nome della nuova Dashboard che l'utente vuole creare;

**- DSLIS: string**

Questo parametro corrisponde al codice che compone la nuova Dashboard scritta dall'utente;

**- ownerEmail: string**

Questo parametro corrisponde all'email dell'autore della Dashboard.

**+editDashboard(dashboardID: number, dashboardName: string, DSLIS: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'identificativo di una Dashboard, il suo nome e il nuovo codice che la compone, utili a modificare una Dashboard già esistente. Successivamente, riceve una risposta dal back-end contenente un errore o la conferma di successo dell'operazione;

**- dashboardID: number**

Questo parametro corrisponde all'identificativo di una Dashboard oggetto della modifica;

**- dashboardName: string**

Questo parametro corrisponde al nuovo nome della Dashboard oggetto della modifica;

– **DSLIS: string**

Questo parametro corrisponde al nuovo codice della Dashboard oggetto della modifica.

**+deleteDashboard(dashboardID: number): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'identificativo di una Dashboard. Successivamente, riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

– **dashboardID: number**

Questo parametro rappresenta l'identificativo della Dashboard che l'utente vuole eliminare;

**+searchDashboard(value: string, email: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente una stringa generica necessaria alla ricerca di una lista di Dashboard e l'email relativa all'utente che esegue la ricerca. Successivamente riceve una risposta dal back-end contenente un errore, oppure la lista composta dalle Dashboard che corrispondono alla ricerca effettuata;

– **value: string**

Questo parametro corrisponde a una generica stringa necessaria a cercare una serie di Dashboard;

– **email: string**

Questo parametro corrisponde a all'indirizzo email dell'utente che desidera effettuare una ricerca.

**+executeDashboard(dashboardID: number): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente un identificativo di una Dashboard, necessario ad eseguirla. Successivamente riceve una risposta dal back-end contenente il risultato dell'operazione in formato JSON;

– **dashboardID: number**

Questo parametro rappresenta l'identificativo della Dashboard che l'utente vuole eseguire.

**+exportDSLIS(dashboardID: number): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente un identificativo di una Dashboard, necessario a esportare il DSLIS corrispondente alla Dashboard indicata. Successivamente riceve una risposta dal back-end contenente il risultato dell'operazione in formato JSON;

– **dashboardID: number**

Questo parametro rappresenta l'identificativo della Dashboard oggetto dell'operazione;

**+retrieveDashboardDSLIS(dashboardID: number): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'identificativo di una Dashboard, utile a ritornare il codice della stessa. Successivamente, riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

– **dashboardID: number**

Questo parametro rappresenta l'identificativo della Dashboard del quale l'utente vuole il codice di cui è composta.

```
+importDSLIS(DSLISEntity: string, DSLISFile: file): void
```

Questo metodo si occupa di inviare una richiesta HTTP, contenente una stringa che rappresenta il tipo di DSLIS che si vuole importare e il file che contiene il DSLIS da importare. Successivamente riceve una risposta dal back-end contenente un errore o il risultato dell'operazione in formato JSON;

- **DSLISEntity: string**

Questo parametro rappresenta il tipo di DSLIS che l'utente desidera importare;

- **DSLISFile: file**

Questo parametro rappresenta il file che contiene il DSLIS che si desidera importare.

## CellWebAPIUtils

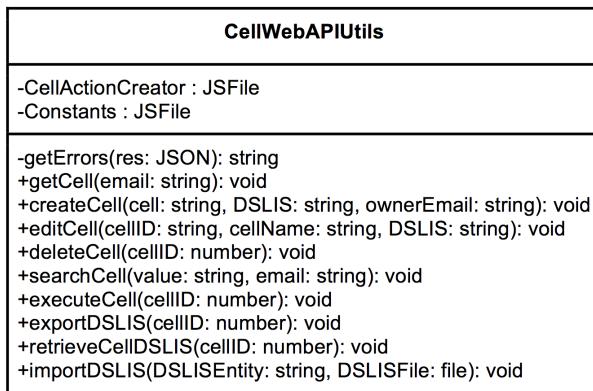


Figura 125: Diagramma della classe CellWebAPIUtils

### Descrizione

Classe che fornisce i metodi per inviare richieste e ricevere risposte nell'ambito delle operazioni sulle Cell create mediante i DSLIS, in modo da permettere al client di comunicare con il server.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per fornire dei metodi utili ad inviare le richieste HTTP verso le API fornite dal server e riceverne le risposte da parte di esso, nell'ambito delle operazioni sulle Cell. Le risposte ricevute saranno inoltrate verso la classe CellActionCreator, che andrà a creare una Action corrispondente.

### Classi ereditate

- React::Superagent

## Relazioni con altri classi

- Front-end::Constants

## Attributi

**-Constants : JSFile**

Questo attributo rappresenta un riferimento al file constants.js;

**-CellActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CellActionCreator.js.

## Metodi

**-getErrors(res: JSON): string**

Questo metodo si occupa di gestire gli errori provenienti dal server. Viene utilizzato dalle funzioni della classe di appartenenza che gestiscono le varie richieste;

**- res: JSON**

Questo parametro corrisponde all'oggetto JSON, rappresentante la risposta del server ad una richiesta;

**+getCell(email: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'indirizzo email di un utente, necessario alla ricerca della lista delle Cell in suo possesso. Successivamente, riceve una risposta dal back-end contenente un errore, oppure la lista composta dalle Cell in possesso dello stesso;

**- email: string**

Questo parametro corrisponde all'email dell'utente del quale si vogliono cercare le Cell, in particolare è lo stesso utente che provoca l'invocazione del metodo.

**+createCell(cellName: string, DSLIS: string, ownerEmail: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente il nome della nuova Cell, il codice che la compone (DSLIS) e l'email dell'autore, con lo scopo di creare una nuova Cell. Successivamente, riceve una risposta dal back-end contenente un errore o la conferma di successo dell'operazione;

**- cellName: string**

Questo parametro corrisponde al nome della nuova Cell che l'utente vuole creare;

**- DSLIS: string**

Questo parametro corrisponde al codice che compone la nuova Cell scritta dall'utente;

**- ownerEmail: string**

Questo parametro corrisponde all'email dell'autore della Cell.

**+editCell(cellID: number, cellName: string, DSLIS: string): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'identificativo di una Cell, il suo nome e il nuovo codice che la compone, utili a modificare una Cell già esistente. Successivamente, riceve una risposta dal back-end contenente un errore o la conferma di successo dell'operazione;

**- `cellID: number`**

Questo parametro corrisponde all'identificativo di una Cell oggetto della modifica;

**- `cellName: string`**

Questo parametro corrisponde al nuovo nome della Cell oggetto della modifica;

**- `DSLIS: string`**

Questo parametro corrisponde al nuovo codice della Cell oggetto della modifica.

**+`deleteCell(cellID: number): void`**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'identificativo di una Cell. Successivamente, riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

**- `cellID: number`**

Questo parametro rappresenta l'identificativo della Cell che l'utente vuole eliminare;

**+`searchCell(value: string, email: string): void`**

Questo metodo si occupa di inviare una richiesta HTTP, contenente una stringa generica necessaria alla ricerca di una lista di Cell e l'email relativa all'utente che esegue la ricerca. Successivamente riceve una risposta dal back-end contenente un errore, oppure la lista composta dalle Cell che corrispondono alla ricerca effettuata;

**- `value: string`**

Questo parametro corrisponde a una generica stringa necessaria a cercare una serie di Cell;

**- `email: string`**

Questo parametro corrisponde a all'indirizzo email dell'utente che desidera effettuare una ricerca.

**+`executeCell(cellID: number): void`**

Questo metodo si occupa di inviare una richiesta HTTP, contenente un identificativo di una Cell, necessario ad eseguirla. Successivamente riceve una risposta dal back-end contenente il risultato dell'operazione in formato JSON;

**- `cellID: number`**

Questo parametro rappresenta l'identificativo della Cell che l'utente vuole eseguire.

**+`exportDSLIS(cellID: number): void`**

Questo metodo si occupa di inviare una richiesta HTTP, contenente un identificativo di una Cell, necessario a esportare il DSLIS corrispondente alla Cell indicata. Successivamente riceve una risposta dal back-end contenente il risultato dell'operazione in formato JSON;

**- `cellID: number`**

Questo parametro rappresenta l'identificativo della Cell oggetto dell'operazione;

**+`retrieveCellDSLIS(cellID: number): void`**

Questo metodo si occupa di inviare una richiesta HTTP, contenente l'identificativo di una Cell, utile a ritornare il codice della stessa. Successivamente, riceve una risposta dal back-end contenente una conferma di successo dell'operazione;

– **cellID: number**

Questo parametro rappresenta l'identificativo della Cell del quale l'utente vuole il codice di cui è composto.

**+importDSLIS(DSLISEntity: string, DSLISFile: file): void**

Questo metodo si occupa di inviare una richiesta HTTP, contenente una stringa che rappresenta il tipo di DSLIS che si vuole importare e il file che contiene il DSLIS da importare. Successivamente riceve una risposta dal back-end contenente un errore o il risultato dell'operazione in formato JSON;

– **DSLISEntity: string**

Questo parametro rappresenta il tipo di DSLIS che l'utente desidera importare;

– **DSLISFile: file**

Questo parametro rappresenta il file che contiene il DSLIS che si desidera importare.

#### 6.1.6 Front-end::Stores

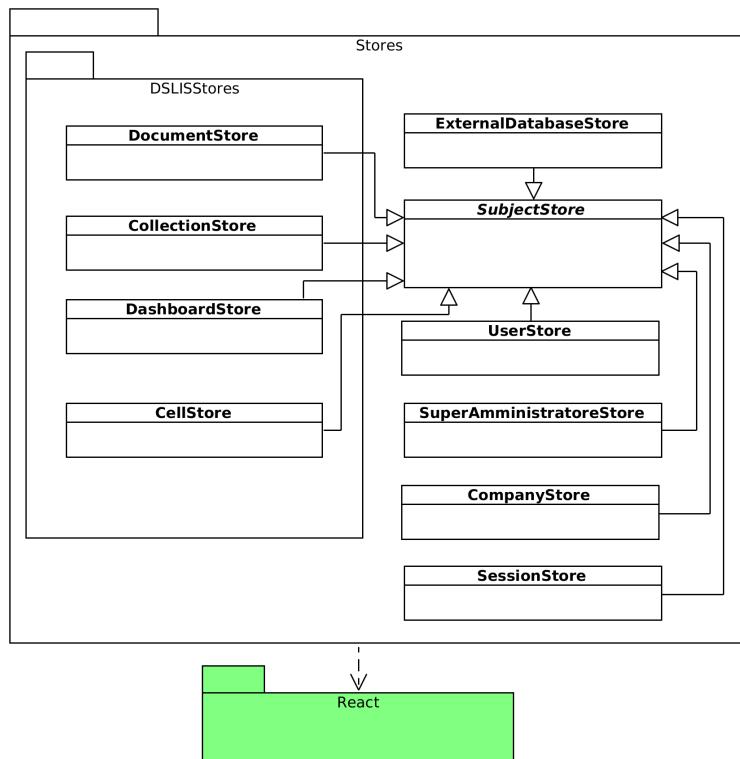


Figura 126: Diagramma delle classi del package Stores

##### 6.1.6.1 Informazioni sul package

###### Descrizione

Package contenente le classi che rappresentano i contenitori che contengono la logica dell'applicazione e il relativo stato. Esse ricevono le Action inoltrate da parte del Dispatcher e ne utilizzano i dati per aggiornare le View.

### Package contenuti

- DSLISStores

### Framework esterni

- React

#### 6.1.6.2 Classi

##### SubjectStore

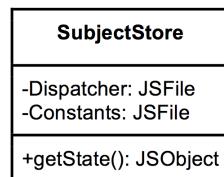


Figura 127: Diagramma della classe SubjectStore

##### Descrizione

Classe che rappresenta la componente Subject del design pattern Observer<sup>6</sup>.

##### Utilizzo

Viene utilizzata per esporre l'interfaccia che consente agli osservatori di iscriversi e cancellarsi.

##### Relazioni con altre classi

- Front-end::Views::ObserverView

##### Estensioni

- Front-end::Stores::ExternalDatabaseStore
- Front-end::Stores::CompanyStore
- Front-end::Stores::SuperAmministratoreStore
- Front-end::Stores::UserStore
- Front-end::Stores::SessionStore

<sup>6</sup>Si veda appendice A.3.1 per approfondimenti

- Front-end::Stores::DSLISStores::DocumentStore
- Front-end::Stores::DSLISStores::CollectionStore
- Front-end::Stores::DSLISStores::DashboardStore
- Front-end::Stores::DSLISStores::CellStore

### UserStore

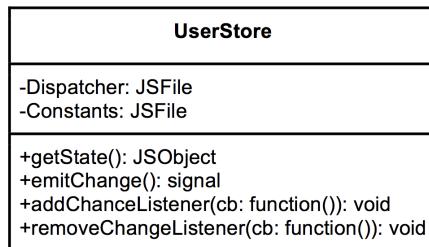


Figura 128: Diagramma della classe UserStore

### Descrizione

Classe che contiene i dati relativi agli utenti.  
 Rappresenta la componente ConcreteSubject del design pattern Observer.  
 Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per contenere la logica dell'applicazione riguardante l'ambito utente. Essa riceve le Action create dall'UserActionCreator ed inoltrate da parte del Dispatcher e successivamente le utilizza per aggiornare la View corrispondente.

### Classi ereditate

- Front-end::Stores::SubjectStore.

### Relazioni con altri classi

- Front-end::Constants
- Front-end::Dispatcher

### Attributi

**Constants : JSFile**

Questo attributo rappresenta un riferimento al file Constants.js;

**Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file Dispatcher.js.

## Metodi

**+getState() : JSONObject**

Questo metodo si occupa di inviare un oggetto contenente gli aggiornamenti delle view;

**+emitChange() : signal**

Questo metodo si occupa di inviare un segnale alle view che notifica un suo cambiamento di stato;

**+addChangeListener(cb : function()) : void**

Questo metodo si occupa di registrare un componente alla medesima store, in modo tale da notificare eventuali cambiamenti;

**cb : function()**

Questo parametro rappresenta una funzione che risponde alla notifica del cambiamento;

**+removeChangeListener(cb: function()) : void**

Questo metodo si occupa di rimuovere una componente dalla medesima store, in modo tale da escluderla dalla notifica di eventuali cambiamenti;

**cb : function()**

Questo parametro rappresenta una funzione che risponde alla notifica del cambiamento.

## SessionStore

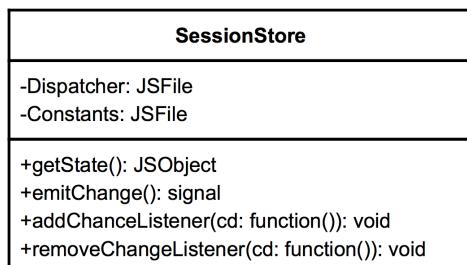


Figura 129: Diagramma della classe SessionStore

## Descrizione

Classe contenente i dati relativi alla sessione dell'utente.

Rappresenta la componente ConcreteSubject del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

## Utilizzo

Viene utilizzata per contenere la logica dell'applicazione riguardante la sessione. Essa riceve le Action create dalla classe SessionActionCreator ed inoltrate da parte del Dispatcher e successivamente le utilizza per aggiornare la View corrispondente.

### Classi ereditate

- Front-end::Stores::SubjectStore.

### Relazioni con altri classi

- Front-end::Constants
- Front-end::Dispatcher

### Attributi

**Constants : JSFile**

Questo attributo rappresenta un riferimento al file Constants.js;

**Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file Dispatcher.js.

### Metodi

**+getState() : JSONObject**

Questo metodo si occupa di inviare un oggetto contenente gli aggiornamenti delle view;

**+emitChange() : signal**

Questo metodo si occupa di inviare un segnale alle view che notifica un suo cambiamento di stato;

**+addChangeListener(cb : function()) : void**

Questo metodo si occupa di registrare un componente alla medesima store, in modo tale da notificare eventuali cambiamenti;

**cb : function()**

Questo parametro rappresenta una funzione che risponde alla notifica del cambiamento;

**+removeChangeListener(cb: function()) : void**

Questo metodo si occupa di rimuovere una componente dalla medesima store, in modo tale da escluderla dalla notifica di eventuali cambiamenti;

**cb : function()**

Questo parametro rappresenta una funzione che risponde alla notifica del cambiamento.

### SuperAmministratoreStore

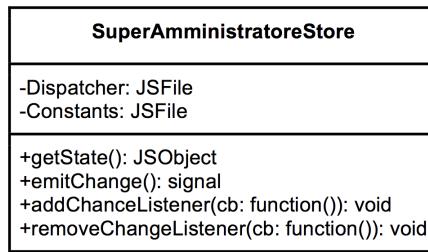


Figura 130: Diagramma della classe SuperAmministratoreStore

### Descrizione

Classe che contiene i dati relativi al Super Amministratore.  
Rappresenta la componente ConcreteSubject del design pattern Observer.  
Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per contenere la logica dell'applicazione riguardante l'ambito Super Amministratore. Essa riceve le Action create dalla classe SuperAmministratoreActionCreator ed inoltrate da parte del Dispatcher e successivamente le utilizza per aggiornare la View corrispondente.

### Classi ereditate

- Front-end::Stores::SubjectStore.

### Relazioni con altri classi

- Front-end::Constants
- Front-end::Dispatcher

### Attributi

**Constants : JSFile**

Questo attributo rappresenta un riferimento al file Constants.js;

**Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file Dispatcher.js.

### Metodi

**+getState() : JSObject**

Questo metodo si occupa di inviare un oggetto contenente gli aggiornamenti delle view;

**+emitChange() : signal**

Questo metodo si occupa di inviare un segnale alle view che notifica un suo cambiamento di stato;

**+addChangeListener(cb : function()) : void**

Questo metodo si occupa di registrare un componente alla medesima store, in modo tale da notificare eventuali cambiamenti;

**cb : function()**

Questo parametro rappresenta una funzione che risponde alla notifica del cambiamento;

**+removeChangeListener(cb: function()) : void**

Questo metodo si occupa di rimuovere una componente dalla medesima store, in modo tale da escluderla dalla notifica di eventuali cambiamenti;

**cb : function()**

Questo parametro rappresenta una funzione che risponde che risponde alla notifica del cambiamento.

### ExternalDatabaseStore

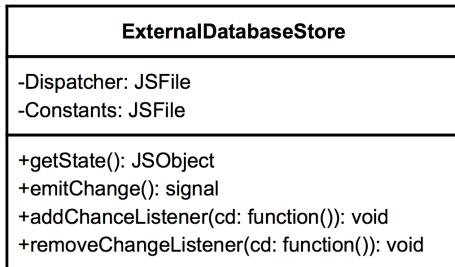


Figura 131: Diagramma della classe ExternalDatabaseStore

### Descrizione

Classe che contiene i dati relativi ai database MongoDB esterni.

Rappresenta la componente ConcreteSubject del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per contenere la logica dell'applicazione riguardante l'ambito dei database MongoDB esterni. Essa riceve le Action create dell'ExternalDatabaseActionCreator ed inoltrate da parte del Dispatcher e successivamente le utilizza per aggiornare la View corrispondente.

### Classi ereditate

- Front-end::Stores::SubjectStore.

### Relazioni con altri classi

- Front-end::Constants
- Front-end::Dispatcher

### Attributi

**Constants : JSFile**

Questo attributo rappresenta un riferimento al file Constants.js;

**Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file Dispatcher.js.

### Metodi

**+getState() : JSObject**

Questo metodo si occupa di inviare un oggetto contenente gli aggiornamenti delle view;

**+emitChange() : signal**

Questo metodo si occupa di inviare un segnale alle view che notifica un suo cambiamento di stato;

**+addChangeListener(cb : function()) : void**

Questo metodo si occupa di registrare un componente alla medesima store, in modo tale da notificare eventuali cambiamenti;

**cb : function()**

Questo parametro rappresenta una funzione che risponde alla notifica del cambiamento;

**+removeChangeListener(cb: function()) : void**

Questo metodo si occupa di rimuovere una componente dalla medesima store, in modo tale da escluderla dalla notifica di eventuali cambiamenti;

**cb : function()**

Questo parametro rappresenta una funzione che risponde alla notifica del cambiamento.

### CompanyStore

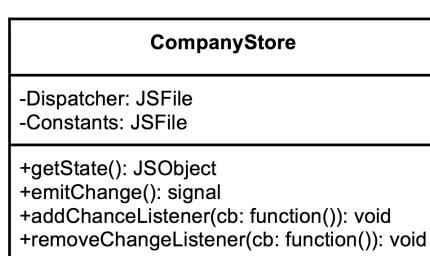


Figura 132: Diagramma della classe CompanyStore

## Descrizione

Classe che contiene i dati relativi alle aziende.  
Rappresenta la componente ConcreteSubject del design pattern Observer.  
Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

## Utilizzo

Viene utilizzata per contenere la logica dell'applicazione riguardante l'ambito delle aziende.  
Essa riceve le Action create dalla classe CompanyActionCreator ed inoltrate da parte del Dispatcher e successivamente le utilizza per aggiornare la View corrispondente.

## Classi ereditate

- Front-end::Stores::SubjectStore.

## Relazioni con altri classi

- Front-end::Constants
- Front-end::Dispatcher

## Attributi

### **Constants : JSFile**

Questo attributo rappresenta un riferimento al file Constants.js;

### **Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file Dispatcher.js.

## Metodi

### **+getState() : JSONObject**

Questo metodo si occupa di inviare un oggetto contenente gli aggiornamenti delle view;

### **+emitChange() : signal**

Questo metodo si occupa di inviare un segnale alle view che notifica un suo cambiamento di stato;

### **+addChangeListener(cb : function()) : void**

Questo metodo si occupa di registrare un componente alla medesima store, in modo tale da notificare eventuali cambiamenti;

### **cb : function()**

Questo parametro rappresenta una funzione che risponde alla notifica del cambiamento;

### **+removeChangeListener(cb: function()) : void**

Questo metodo si occupa di rimuovere una componente dalla medesima store, in modo tale da escluderla dalla notifica di eventuali cambiamenti;

### **cb : function()**

Questo parametro rappresenta una funzione che risponde che risponde alla notifica del cambiamento.

#### 6.1.7 Front-end::Stores::DSLISStores

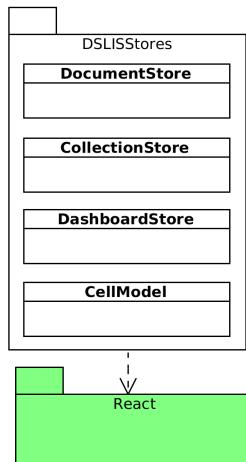


Figura 133: Diagramma delle classi del package DSLISStores

##### 6.1.7.1 Informazioni sul package

**Descrizione** Package contenente le classi che rappresentano i contenitori che contengono la logica dell'applicazione e il relativo stato nell'ambito dei DSLIS.

##### Framework esterni

- React

##### 6.1.7.2 Classi

###### CollectionStore

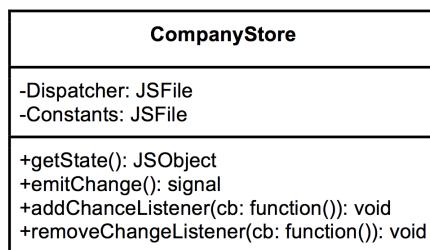


Figura 134: Diagramma della classe CompanyStore

## Descrizione

Classe che contiene i dati relativi alle Collection, create mediante i DSLIS.  
Rappresenta la componente ConcreteSubject del design pattern Observer.  
Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

## Utilizzo

Viene utilizzata per contenere la logica dell'applicazione riguardante l'ambito delle Collection.  
Essa riceve le Action create dalla classe CollectionActionCreator ed inoltrate da parte del Dispatcher e successivamente le utilizza per aggiornare la View corrispondente.

## Classi ereditate

- Front-end::Stores::SubjectStore.

## Relazioni con altri classi

- Front-end::Constants
- Front-end::Dispatcher

## Attributi

### **Constants : JSFile**

Questo attributo rappresenta un riferimento al file Constants.js;

### **Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file Dispatcher.js.

## Metodi

### **+getState() : JSONObject**

Questo metodo si occupa di inviare un oggetto contenente gli aggiornamenti delle view;

### **+emitChange() : signal**

Questo metodo si occupa di inviare un segnale alle view che notifica un suo cambiamento di stato;

### **+addChangeListener(cb : function()) : void**

Questo metodo si occupa di registrare un componente alla medesima store, in modo tale da notificare eventuali cambiamenti;

### **cb : function()**

Questo parametro rappresenta una funzione che risponde alla notifica del cambiamento;

### **+removeChangeListener(cb: function()) : void**

Questo metodo si occupa di rimuovere una componente dalla medesima store, in modo tale da escluderla dalla notifica di eventuali cambiamenti;

### **cb : function()**

Questo parametro rappresenta una funzione che risponde che risponde alla notifica del cambiamento.

## DocumentStore

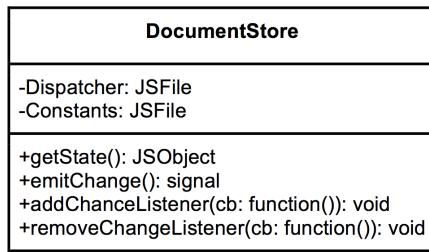


Figura 135: Diagramma della classe DocumentStore

### Descrizione

Classe che contiene i dati relativi ai Document, creati mediante i DSLIS. Rappresenta la componente ConcreteSubject del design pattern Observer. Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per contenere la logica dell'applicazione riguardante l'ambito dei Document. Essa riceve le Action create dalla classe DocumentActionCreator ed inoltrate da parte del Dispatcher e successivamente le utilizza per aggiornare la View corrispondente.

### Classi ereditate

- Front-end::Stores::SubjectStore.

### Relazioni con altri classi

- Front-end::Constants
- Front-end::Dispatcher

### Attributi

#### **Constants : JSFile**

Questo attributo rappresenta un riferimento al file Constants.js;

#### **Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file Dispatcher.js.

## Metodi

**+getState() : JSONObject**

Questo metodo si occupa di inviare un oggetto contenente gli aggiornamenti delle view;

**+emitChange() : signal**

Questo metodo si occupa di inviare un segnale alle view che notifica un suo cambiamento di stato;

**+addChangeListener(cb : function()) : void**

Questo metodo si occupa di registrare un componente alla medesima store, in modo tale da notificare eventuali cambiamenti;

**cb : function()**

Questo parametro rappresenta una funzione che risponde alla notifica del cambiamento;

**+removeChangeListener(cb: function()) : void**

Questo metodo si occupa di rimuovere una componente dalla medesima store, in modo tale da escluderla dalla notifica di eventuali cambiamenti;

**cb : function()**

Questo parametro rappresenta una funzione che risponde alla notifica del cambiamento.

## DashboardStore

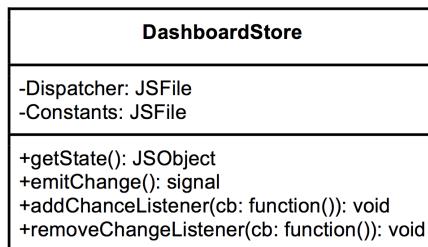


Figura 136: Diagramma della classe DashboardStore

## Descrizione

Classe che contiene i dati relativi alle Dashboard, create mediante i DSLIS.  
Rappresenta la componente ConcreteSubject del design pattern Observer.  
Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

## Utilizzo

Viene utilizzata per contenere la logica dell'applicazione riguardante l'ambito delle Dashboard.  
Essa riceve le Action create dalla classe DashboardActionCreator ed inoltrate da parte del Dispatcher e successivamente le utilizza per aggiornare la View corrispondente.

### Classi ereditate

- Front-end::Stores::SubjectStore.

### Relazioni con altri classi

- Front-end::Constants
- Front-end::Dispatcher

### Attributi

**Constants : JSFile**

Questo attributo rappresenta un riferimento al file Constants.js;

**Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file Dispatcher.js.

### Metodi

**+getState() : JSONObject**

Questo metodo si occupa di inviare un oggetto contenente gli aggiornamenti delle view;

**+emitChange() : signal**

Questo metodo si occupa di inviare un segnale alle view che notifica un suo cambiamento di stato;

**+addChangeListener(cb : function()) : void**

Questo metodo si occupa di registrare un componente alla medesima store, in modo tale da notificare eventuali cambiamenti;

**cb : function()**

Questo parametro rappresenta una funzione che risponde alla notifica del cambiamento;

**+removeChangeListener(cb: function()) : void**

Questo metodo si occupa di rimuovere una componente dalla medesima store, in modo tale da escluderla dalla notifica di eventuali cambiamenti;

**cb : function()**

Questo parametro rappresenta una funzione che risponde alla notifica del cambiamento.

### CellStore

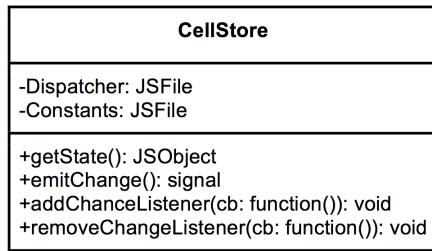


Figura 137: Diagramma della classe CellStore

### Descrizione

Classe che contiene i dati relativi alle Cell, create mediante i DSLIS.  
 Rappresenta la componente ConcreteSubject del design pattern Observer.  
 Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per contenere la logica dell'applicazione riguardante l'ambito delle Cell. Essa riceve le Action create dalla classe CellActionCreator ed inoltrate da parte del Dispatcher e successivamente le utilizza per aggiornare la View corrispondente.

### Classi ereditate

- Front-end::Stores::SubjectStore.

### Relazioni con altri classi

- Front-end::Constants
- Front-end::Dispatcher

### Attributi

#### **Constants : JSFile**

Questo attributo rappresenta un riferimento al file Constants.js;

#### **Dispatcher : JSFile**

Questo attributo rappresenta un riferimento al file Dispatcher.js.

### Metodi

#### **+getState() : JSObject**

Questo metodo si occupa di inviare un oggetto contenente gli aggiornamenti delle view;

#### **+emitChange() : signal**

Questo metodo si occupa di inviare un segnale alle view che notifica un suo cambiamento di stato;

**+addChangeListener(cb : function()) : void**

Questo metodo si occupa di registrare un componente alla medesima store, in modo tale da notificare eventuali cambiamenti;

**cb : function()**

Questo parametro rappresenta una funzione che risponde alla notifica del cambiamento;

**+removeChangeListener(cb: function()) : void**

Questo metodo si occupa di rimuovere una componente dalla medesima store, in modo tale da escluderla dalla notifica di eventuali cambiamenti;

**cb : function()**

Questo parametro rappresenta una funzione che risponde alla notifica del cambiamento.

### 6.1.8 Front-end::Views

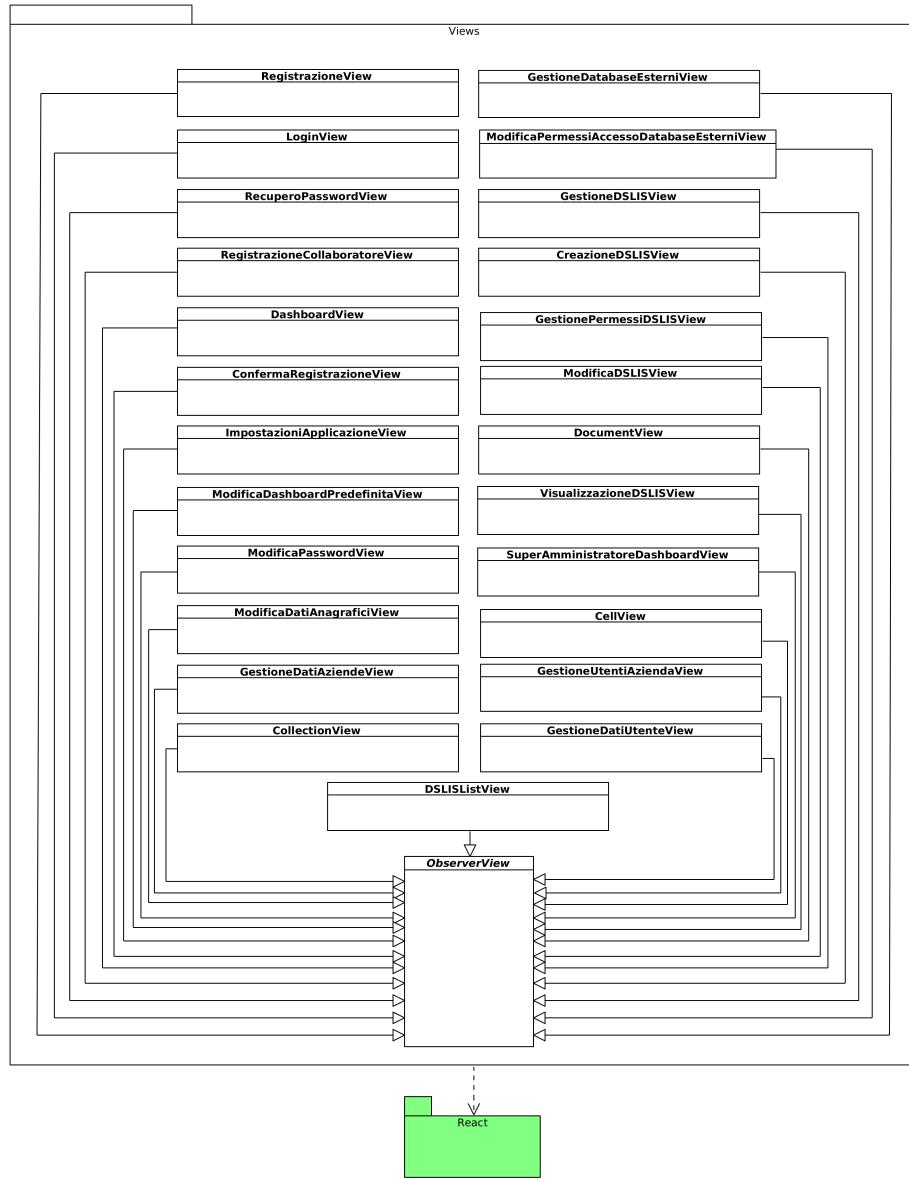


Figura 138: Diagramma delle classi del package Views

#### 6.1.8.1 Informazioni sul package

##### Descrizione

Package comprendente le classi che costituiscono la view del componente Front-end del sistema MaaS. Ogni view rappresenta una pagina HTML con determinati campi, i quali

verranno popolati con i dati richiesti, forniti dal sistema per informare l'utente, o forniti dall'utente per compiere determinate azioni.

### Framework esterni

- React

#### 6.1.8.2 Classi

##### ObserverView

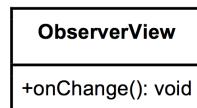


Figura 139: Diagramma della classe ObserverView

##### Descrizione

Classe che rappresenta la componente Observer del design pattern Observer.

##### Utilizzo

Viene utilizzata per esporre l'interfaccia che consente di aggiornare gli osservatori in caso di cambio di stato del soggetto osservato.

##### Estensioni

- Front-end::Views::RegistrazioneView
- Front-end::Views::LoginView
- Front-end::Views::RecuperoPasswordView
- Front-end::Views::GestioneDSLISView
- Front-end::Views::GestioneDatabaseEsterniView
- Front-end::Views::ModificaPermessiAccessoDatabaseEsterniView
- Front-end::Views::RegistrazioneCollaboratoreView
- Front-end::Views::CreazioneDSLISView
- Front-end::Views::ModificaDSLISView
- Front-end::Views::DashboardView
- Front-end::Views::ConfermaRegistrazioneView
- Front-end::Views::GesionePermessiDSLISView
- Front-end::Views::ImpostazioniApplicazioneView

- Front-end::Views::GestioneUtentiAziendaView
- Front-end::Views::ModificaDashboardPredefinitaView
- Front-end::Views::ModificaPasswordView
- Front-end::Views::VisualizzazioneDSLISView
- Front-end::Views::ModificaDatiAnagraficiView
- Front-end::Views::GestioneDatiAziendeView
- Front-end::Views::SuperAmministratoreDashboardView
- Front-end::Views::GestioneDatiUtenteView
- Front-end::Views::DocumentView
- Front-end::Views::DSLISListView
- Front-end::Views::CollectionView
- Front-end::Views::CellView

### Attributi

Assenti

### Metodi

`+onChange(): void`

Questo metodo rappresenta l'interfaccia per gli omonimi metodi delle classi della view concrete.

### RegistrazioneView

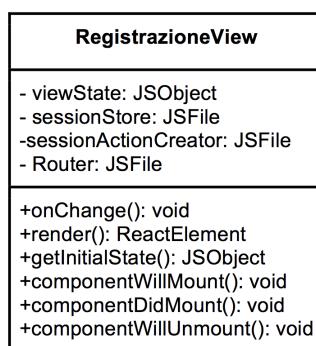


Figura 140: Diagramma della classe RegistrazioneView

## Descrizione

Questa classe descrive la pagina utile a eseguire la registrazione di un nuovo proprietario nel sistema MaaS inserendo negli appositi campi l'email e la password.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

## Utilizzo

Viene utilizzata per mostrare la pagina di registrazione per un nuovo proprietario; quando un utente interagisce con essa, viene chiamato un metodo dell'UserActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dell'UserStore.

## Classi ereditate

- Front-end::Views::ObserverView

## Relazioni con altre classi

- Front-end::Router
- Front-end::Stores::SessionStore
- Front-end::ActionCreators::SessionActionCreator

## Attributi

### **-viewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

### **-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

### **-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

### **-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreator.js.

## Metodi

### **+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState);`

### **+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View)  $DOM_c$  nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

## LoginView

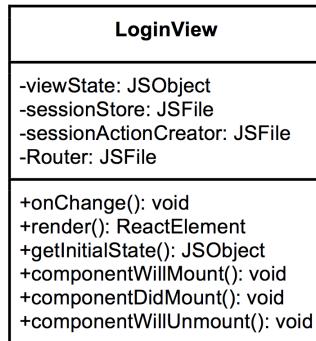


Figura 141: Diagramma della classe LoginView

## Descrizione

Questa classe descrive la pagina utile a eseguire il login di un utente non autenticato mettendo a disposizione di esso un form nella quale inserire email e password; inoltre viene fornito un link grazie al quale viene data la possibilità di recuperare la propria password.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

## Utilizzo

Viene utilizzata per mostrare la pagina di login di un utente non ancora autenticato; quando un utente interagisce con essa, viene chiamato un metodo dell'UserActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dell'UserStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Router
- Front-end::Stores::SessionStore
- Front-end::ActionCreators::SessionActionCreator

### Attributi

**-ViewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreator.js.

### Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati

dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

### RecuperoPasswordView

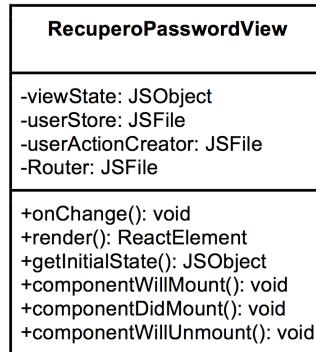


Figura 142: Diagramma della classe RecuperoPasswordView

### Descrizione

Questa classe descrive la pagina utile a consentire all'utente di modificare la propria password in caso essa sia stata smarrita; al fine di eseguire questa operazione verrà fornito un form nel quale dovrà venire inserita la nuova password e la conferma della stessa.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina recupero della password per un utente non ancora autenticato; quando un utente interagisce con essa, viene chiamato un metodo dell'UserActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dell'UserStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Router
- Front-end::Stores::UserStore
- Front-end::ActionCreators::UserActionCreator

## Attributi

**-ViewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js.

## Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

## DashboardView

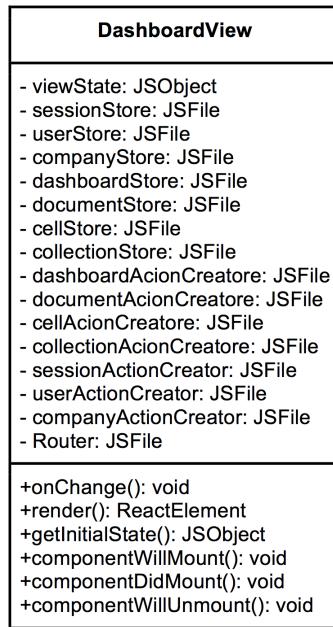


Figura 143: Diagramma della classe DashboardView

### Descrizione

Questa classe descrive la pagina utile a visualizzare la Dashboard, predefinita o personalizzata, propria del profilo dell'utente.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina nella quale si trova la Dashboard; quando un utente interagisce con essa, viene chiamato un metodo dello DashboardActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte del DashboardStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::DSLISStores::DashboardStore
- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore

- Front-end::Stores::DSLISStores::DashboardStore
- Front-end::Stores::DSLISStores::CellStore
- Front-end::Stores::DSLISStores::CollectionStore
- Front-end::Stores::DSLISStores::DocumentStore
- Front-end::Router
- Front-end::ActionCreators::SessionActionCreator
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::DSLISActionCreators::DashboardActionCreator
- Front-end::ActionCreators::DSLISActionCreators::DocumentActionCreator
- Front-end::ActionCreators::DSLISActionCreators::CollectionActionCreator
- Front-end::ActionCreators::DSLISActionCreators::CellActionCreator

### Attributi

- **-ViewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

- **-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

- **-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

- **-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

- **-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

- **-documentStore : JSFile**

Questo attributo rappresenta un riferimento al file DocumentStore.js;

- **-cellStore : JSFile**

Questo attributo rappresenta un riferimento al file CellStore.js;

- **-dashboardStore : JSFile**

Questo attributo rappresenta un riferimento al file DashboardStore.js;

- **-collectionStore : JSFile**

Questo attributo rappresenta un riferimento al file CollectionStore.js;

- **-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

- **-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-cellActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CellActionCreators.js;

**-collectionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CollectionActionCreators.js;

**-dashboardActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file DashboardActionCreators.js;

**-documentActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file DocumentActionCreators.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreators.js.

## Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

## DSLISListView

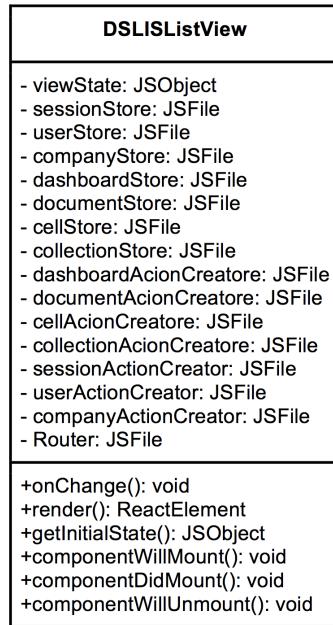


Figura 144: Diagramma della classe DSLISListView

### Descrizione

Questa classe descrive la pagina che visualizza la lista di tutti i DSLIS creati dall'utente o condivisi con lo stesso.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina nella quale si trova la lista dei DSLIS; quando un utente interagisce con essa, viene chiamato un metodo dello DSLISActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte del DSLISStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::DSLISStores::DashboardStore
- Front-end::Stores::DSLISStores::CellStore
- Front-end::Stores::DSLISStores::CollectionStore
- Front-end::Stores::DSLISStores::DocumentStore

- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore
- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator
- Front-end::ActionCreators::DSLISActionCreators::DocumentActionCreator
- Front-end::ActionCreators::DSLISActionCreators::CellActionCreator
- Front-end::ActionCreators::DSLISActionCreators::DashboardActionCreator
- Front-end::ActionCreators::DSLISActionCreators::CollectionActionCreator

### Attributi

**-ViewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-documentStore : JSFile**

Questo attributo rappresenta un riferimento al file DocumentStore.js;

**-cellStore : JSFile**

Questo attributo rappresenta un riferimento al file CellStore.js;

**-dashboardStore : JSFile**

Questo attributo rappresenta un riferimento al file DashboardStore.js;

**-collectionStore : JSFile**

Questo attributo rappresenta un riferimento al file CollectionStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-cellActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CellActionCreators.js;

**-collectionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CollectionActionCreators.js;

**-dashboardActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file DashboardActionCreators.js;

**-documentActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file DocumentActionCreators.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreators.js.

## Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

## RegistrazioneCollaboratoreView

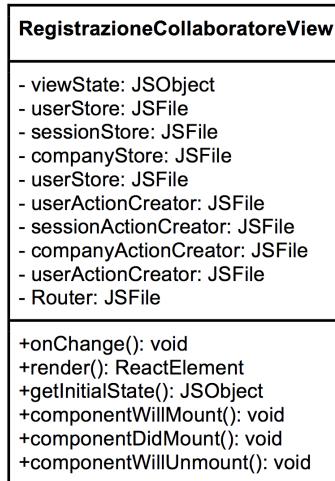


Figura 145: Diagramma della classe RegistrazioneCollaboratoreView

### Descrizione

Questa classe descrive la pagina che consente ad un utente invitato di potersi registrare al sistema fornendo un form nel quale potrà inserire una password che utilizzerà per accedere a esso.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina di registrazione di un nuovo collaboratore; quando un utente interagisce con essa, viene chiamato un metodo dello UserActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dello UserStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore
- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator

- Front-end::ActionCreators::SessionActionCreator

## Attributi

**-ViewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-userActionCreators : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreators.js;

**-companyActionCreators : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreators.js;

**-sessionActionCreators : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreators.js.

## Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come

ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

### ConfermaRegistrazioneView

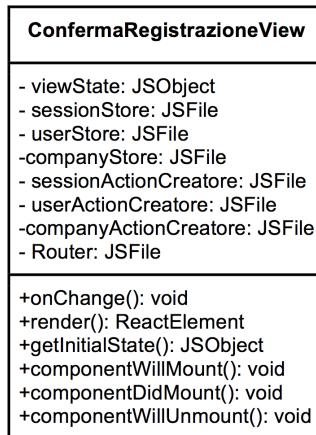


Figura 146: Diagramma della classe ConfermaRegistrazioneView

### Descrizione

Questa classe descrive la prima pagina che l'utente appena registrato visualizza, essa gli confermerà l'avvenuta registrazione e conterrà un link che gli permetterà di entrare nella propria Dashboard.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina di conferma dell'avvenuta registrazione di un nuovo collaboratore; quando un utente interagisce con essa, viene chiamato un metodo dello UserActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dello UserStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore

- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator

### Attributi

**-viewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreator.js.

### Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

### ImpostazioniApplicazioneView

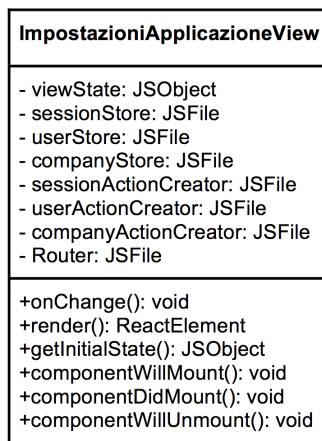


Figura 147: Diagramma della classe ImpostazioniApplicazioneView

### Descrizione

Questa classe descrive la pagina che permette all'utente di modificare alcune impostazioni dell'applicazione.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina che contiene le impostazioni dell'applicazione; quando un utente interagisce con essa, viene chiamato un metodo dello UserActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dello UserStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore
- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator

### Attributi

**-ViewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreator.js.

### Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

### ModificaDashboardPredefinitaView

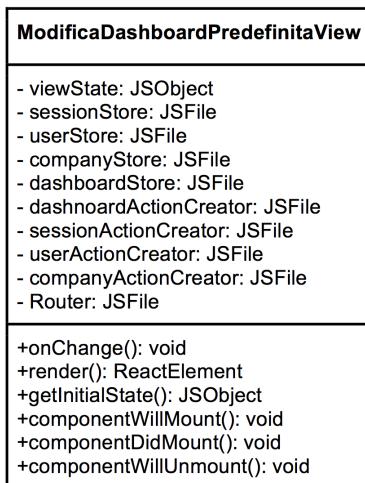


Figura 148: Diagramma della classe ModificaDashboardPredefinitaView

### Descrizione

Questa classe descrive la pagina che permette all'utente di modificare la preferenza riguardo alla scelta della Dashboard attiva, fornendo la lista delle Dashboard presenti e un bottone che permette la conferma dell'operazione.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina nella quale vi sono le funzioni che consentono di modificare la Dashboard che andrà a sostituire quella attualmente predefinita; quando un utente interagisce con essa, viene chiamato un metodo dello DashboardActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dello DashboardStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::DSLISStores::DashboardStore
- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore
- Front-end::Stores::DSLISStores::DashboardStore
- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator
- Front-end::ActionCreators::DSLISActionCreators::DashboardActionCreator

### Attributi

**-viewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**dashboardStore : JSFile**

Questo attributo rappresenta un riferimento al file DashboardStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-dashboardActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file DashboardActionCreator.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreator.js.

## Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

## ModificaDatiAnagraficiView

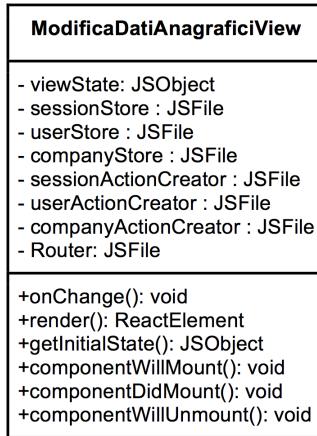


Figura 149: Diagramma della classe ModificaDatiAnagraficiView

### Descrizione

Questa classe descrive la pagina che consente di modificare i dati anagrafici dell'utente fornendo form che gli consentono, in particolare, di variare nome, cognome, data di nascita e sesso.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina di modifica dei dati anagrafici dell'utente autenticato; quando un utente interagisce con essa, viene chiamato un metodo dello UserActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dello UserStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore
- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator

## Attributi

**-ViewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreator.js.

## Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati

dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

### ModificaPasswordView

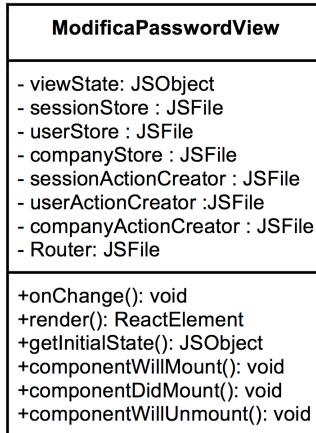


Figura 150: Diagramma della classe ModificaPasswordView

### Descrizione

Questa classe descrive la pagina che consente all'utente di modificare la propria password fornendo un form nel quale potrà inserire la password attuale, la nuova password e la conferma della stessa.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina di modifica della password dell'utente autenticato; quando un utente interagisce con essa, viene chiamato un metodo dello UserActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dello UserStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore

- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator

### Attributi

**-viewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreator.js.

### Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

### GestioneDatabaseEsterniView

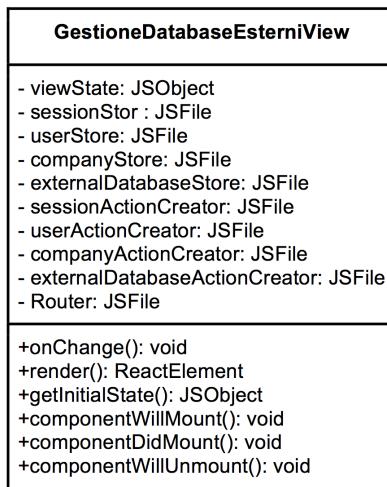


Figura 151: Diagramma della classe GestioneDatabaseEsterniView

### Descrizione

Questa classe descrive la pagina che consente all'utente di eseguire una serie di operazioni quali: connettere un proprio database al sistema MaaS fornendo un form nel quale potrà inserire il nome e l'indirizzo del nuovo database, disconnettere un database precedentemente importato mediante selezione dello stesso e conferma dell'operazione di eliminazione; inoltre mantiene un collegamento con la pagina utile a gestire i permessi di accesso ai database da parte di altri utenti.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina di gestione dei database esterni; quando un utente interagisce con essa, viene chiamato un metodo dello ExternalDataBaseActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dello ExternalDatabaseStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore
- Front-end::Stores::ExternalDatabaseStore
- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator
- Front-end::ActionCreators::ExternalDatabaseActionCreator

### Attributi

**-viewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-externalDatabaseStore : JSFile**

Questo attributo rappresenta un riferimento al file ExternalDatabaseStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-externalDatabaseActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file ExternalDatabaseActionCreator.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreator.js.

## Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

## ModificaPermessiAccessoDatabaseEsterniView

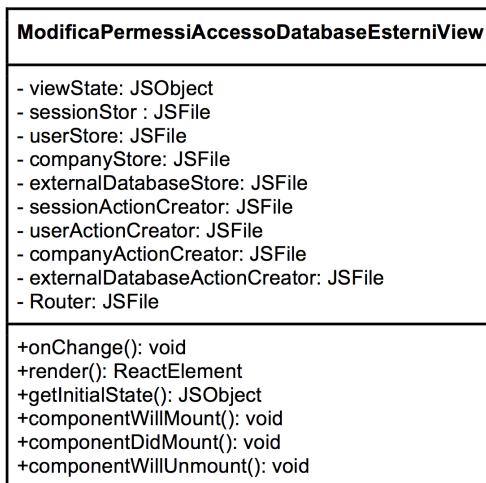


Figura 152: Diagramma della classe ModificaPermessiAccessoDatabaseEsterniView

## Descrizione

Questa classe descrive la pagina che consente all'utente di modificare i permessi di accesso ai database esterni fornendo, in particolare, la lista dei database presenti, degli utenti dell'azienda e dei radio button che consentono di selezionare il livello di permesso desiderato. Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

## Utilizzo

Viene utilizzata per mostrare la pagina di gestione dei permessi di accesso ai database esterni; quando un utente interagisce con essa, viene chiamato un metodo dello ExternalDataBaseActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dello ExternalDatabaseStore.

## Classi ereditate

- Front-end::Views::ObserverView

## Relazioni con altre classi

- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore
- Front-end::Stores::ExternalDatabaseStore
- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator
- Front-end::ActionCreators::ExternalDatabaseActionCreator

## Attributi

### **-viewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

### **-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

### **-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

### **-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-externalDatabaseStore : JSFile**

Questo attributo rappresenta un riferimento al file ExternalDatabaseStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-externalDatabaseActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file ExternalDatabaseActionCreator.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreator.js.

## Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

## GestioneDSLISView



Figura 153: Diagramma della classe GestioneDSLISView

### Descrizione

Questa classe descrive la pagina che consente all'utente di visualizzare tutti i DLSIS prodotti dallo stesso o condivisi con lui, sui quali possiede determinati permessi, fornendo bottoni che consentono all'utente di eseguire operazioni quali: creare un nuovo DSLIS, modificare, eliminare, visualizzare il set di istruzioni che compone un DSLIS o eseguirne una esistente.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina di gestione dei DSLIS; quando un utente interagisce con essa, viene chiamato un metodo dello DSLISActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dello DSLISStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::DSLISStores::DashboardStore
- Front-end::Stores::DSLISStores::CellStore

- Front-end::Stores::DSLISStores::CollectionStore
- Front-end::Stores::DSLISStores::DocumentStore
- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore
- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator
- Front-end::ActionCreators::DSLISActionCreators::DocumentActionCreator
- Front-end::ActionCreators::DSLISActionCreators::CellActionCreator
- Front-end::ActionCreators::DSLISActionCreators::DashboardActionCreator
- Front-end::ActionCreators::DSLISActionCreators::CollectionActionCreator

### Attributi

**-ViewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-documentStore : JSFile**

Questo attributo rappresenta un riferimento al file DocumentStore.js;

**-cellStore : JSFile**

Questo attributo rappresenta un riferimento al file CellStore.js;

**-dashboardStore : JSFile**

Questo attributo rappresenta un riferimento al file DashboardStore.js;

**-collectionStore : JSFile**

Questo attributo rappresenta un riferimento al file CollectionStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-cellActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CellActionCreator.js;

**-collectionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CollectionActionCreator.js;

**-dashboardActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file DashboardActionCreator.js;

**-documentActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file DocumentActionCreator.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreator.js.

## Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

## GestionePermessiDSLISView

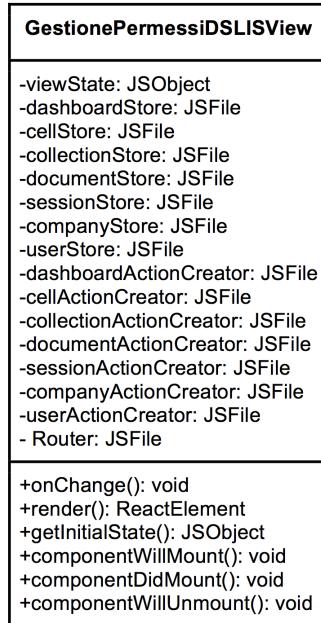


Figura 154: Diagramma della classe GestionePermessiDSLISView

### Descrizione

Questa classe descrive la pagina che consente all'utente di assegnare o togliere i permessi di esecuzione, lettura e scrittura di un DSLIS a un altro utente.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina di gestione dei permessi di esecuzione, lettura e scrittura; quando un utente interagisce con essa, viene chiamato un metodo dello DSLISActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dello DSLISStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::DSLISStores::DashboardStore
- Front-end::Stores::DSLISStores::CellStore
- Front-end::Stores::DSLISStores::CollectionStore
- Front-end::Stores::DSLISStores::DocumentStore

- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore
- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator
- Front-end::ActionCreators::DSLISActionCreators::DocumentActionCreator
- Front-end::ActionCreators::DSLISActionCreators::CellActionCreator
- Front-end::ActionCreators::DSLISActionCreators::DashboardActionCreator
- Front-end::ActionCreators::DSLISActionCreators::CollectionActionCreator

### Attributi

**-ViewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-documentStore : JSFile**

Questo attributo rappresenta un riferimento al file DocumentStore.js;

**-cellStore : JSFile**

Questo attributo rappresenta un riferimento al file CellStore.js;

**-dashboardStore : JSFile**

Questo attributo rappresenta un riferimento al file DashboardStore.js;

**-collectionStore : JSFile**

Questo attributo rappresenta un riferimento al file CollectionStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-cellActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CellActionCreators.js;

**-collectionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CollectionActionCreators.js;

**-dashboardActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file DashboardActionCreators.js;

**-documentActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file DocumentActionCreators.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreators.js.

## Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

## CreazioneDSLISView

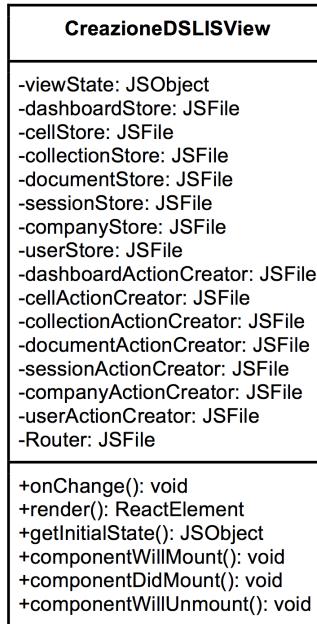


Figura 155: Diagramma della classe CreazioneDSLISView

### Descrizione

Questa classe descrive la pagina utile a fornire strumenti che consentano all'utente di creare un proprio DSLIS, in particolare fornendo un editor di testo e bottoni che permettano ad esempio di salvare il lavoro eseguito.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina di creazione di un nuovo DSLIS; quando un utente interagisce con essa, viene chiamato un metodo dello DSLISActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dello DSLISStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::DSLISStores::DashboardStore
- Front-end::Stores::DSLISStores::CellStore
- Front-end::Stores::DSLISStores::CollectionStore

- Front-end::Stores::DSLISStores::DocumentStore
- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore
- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator
- Front-end::ActionCreators::DSLISActionCreators::DocumentActionCreator
- Front-end::ActionCreators::DSLISActionCreators::CellActionCreator
- Front-end::ActionCreators::DSLISActionCreators::DashboardActionCreator
- Front-end::ActionCreators::DSLISActionCreators::CollectionActionCreator

### Attributi

**-ViewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-documentStore : JSFile**

Questo attributo rappresenta un riferimento al file DocumentStore.js;

**-cellStore : JSFile**

Questo attributo rappresenta un riferimento al file CellStore.js;

**-dashboardStore : JSFile**

Questo attributo rappresenta un riferimento al file DashboardStore.js;

**-collectionStore : JSFile**

Questo attributo rappresenta un riferimento al file CollectionStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-cellActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CellActionCreators.js;

**-collectionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CollectionActionCreators.js;

**-dashboardActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file DashboardActionCreators.js;

**-documentActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file DocumentActionCreators.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreators.js.

## Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

## GestioneDSLISView

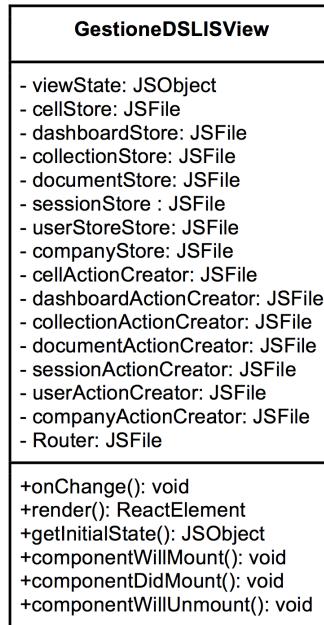


Figura 156: Diagramma della classe GestioneDSLISView

**Descrizione** Questa classe descrive la pagina che consente all'utente di visualizzare le opzioni utili a:

- Gestire i permessi di
- Gestire i permessi di scrittura dei DSLIS in particolare fornendo una lista degli stesse, degli utenti del sistema facenti parte della medesima azienda dell'utente al quale si intende modificare i permessi e un checkBox che consente di attribuire o meno il permesso in oggetto;
- Gestire i permessi di lettura dei DSLIS in particolare fornendo una lista degli stesse, degli utenti del sistema facenti parte della medesima azienda dell'utente che al quale si intende modificare i permessi e un checkBox che consente di attribuire o meno il permesso in oggetto.

Rappresenta la componente ConcreteObserver del design pattern Observer.

**Utilizzo** Viene utilizzata per mostrare la pagina di gestione dei DSLIS; quando un utente interagisce con essa, viene chiamato un metodo dello DSLISActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dello DSLISStore.

### ModificaDSLISView

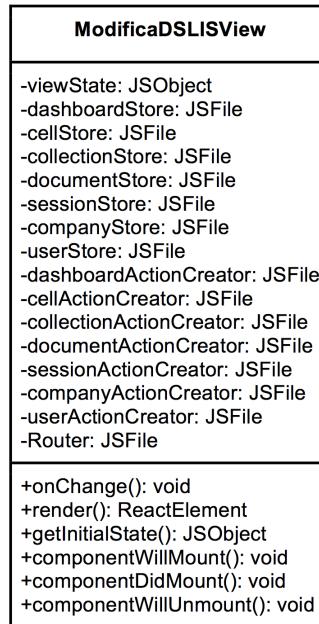


Figura 157: Diagramma della classe ModificaDSLISView

### Descrizione

Questa classe descrive la pagina che consente all'utente di modificare un DSLIS in suo possesso, in particolare fornendo un editor testuale nel quale si troverà il DSLIS. Rappresenta la componente ConcreteObserver del design pattern Observer. Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina di modifica di un DSLIS; quando un utente interagisce con essa, viene chiamato un metodo dello DSLISActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dello DSLISStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::DSLISStores::DashboardStore
- Front-end::Stores::DSLISStores::CellStore
- Front-end::Stores::DSLISStores::CollectionStore
- Front-end::Stores::DSLISStores::DocumentStore

- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore
- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator
- Front-end::ActionCreators::DSLISActionCreators::DocumentActionCreator
- Front-end::ActionCreators::DSLISActionCreators::CellActionCreator
- Front-end::ActionCreators::DSLISActionCreators::DashboardActionCreator
- Front-end::ActionCreators::DSLISActionCreators::CollectionActionCreator

### Attributi

**-viewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-documentStore : JSFile**

Questo attributo rappresenta un riferimento al file DocumentStore.js;

**-cellStore : JSFile**

Questo attributo rappresenta un riferimento al file CellStore.js;

**-dashboardStore : JSFile**

Questo attributo rappresenta un riferimento al file DashboardStore.js;

**-collectionStore : JSFile**

Questo attributo rappresenta un riferimento al file CollectionStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-cellActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CellActionCreators.js;

**-collectionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CollectionActionCreators.js;

**-dashboardActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file DashboardActionCreators.js;

**-documentActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file DocumentActionCreators.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreators.js.

## Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

## VisualizzazioneDSLISView

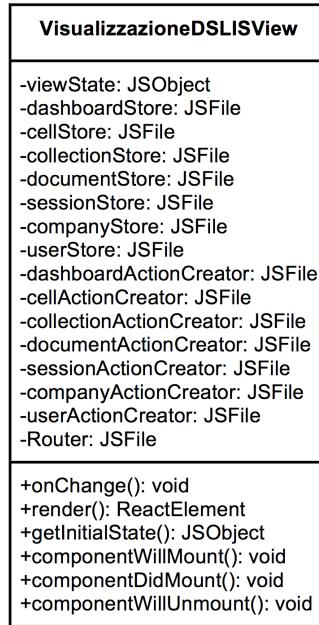


Figura 158: Diagramma della classe VisualizzazioneDSLISView

### Descrizione

Questa classe descrive la pagina utile a visualizzare il DSLIS in oggetto senza che essa possa venire modificata.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina di visualizzazione di un DSLIS; quando un utente interagisce con essa, viene chiamato un metodo dello DSLISActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dello DSLISStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::DSLISStores::DashboardStore
- Front-end::Stores::DSLISStores::CellStore
- Front-end::Stores::DSLISStores::CollectionStore
- Front-end::Stores::DSLISStores::DocumentStore

- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore
- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator
- Front-end::ActionCreators::DSLISActionCreators::DocumentActionCreator
- Front-end::ActionCreators::DSLISActionCreators::CellActionCreator
- Front-end::ActionCreators::DSLISActionCreators::DashboardActionCreator
- Front-end::ActionCreators::DSLISActionCreators::CollectionActionCreator

### Attributi

**-ViewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-documentStore : JSFile**

Questo attributo rappresenta un riferimento al file DocumentStore.js;

**-cellStore : JSFile**

Questo attributo rappresenta un riferimento al file CellStore.js;

**-dashboardStore : JSFile**

Questo attributo rappresenta un riferimento al file DashboardStore.js;

**-collectionStore : JSFile**

Questo attributo rappresenta un riferimento al file CollectionStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-cellActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CellActionCreators.js;

**-collectionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CollectionActionCreators.js;

**-dashboardActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file DashboardActionCreators.js;

**-documentActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file DocumentActionCreators.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreators.js.

## Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

## CollectionView

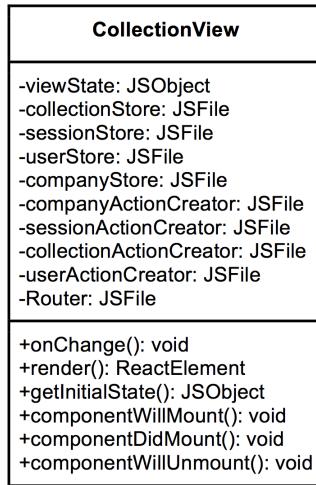


Figura 159: Diagramma della classe CollectionView

### Descrizione

La classe descrivere la pagina che visualizza i documenti della Collection selezionata.  
 Rappresenta la componente ConcreteObserver del design pattern Observer.  
 Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina di visualizzazione di una Collection; quando un utente interagisce con essa, viene chiamato un metodo del CollectionActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte del CollectionStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::DSLISStores::CollectionStore
- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore
- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator

- Front-end::ActionCreators::SessionActionCreator
- Front-end::ActionCreators::DSLISActionCreators::CollectionActionCreator

### Attributi

**-viewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-collectionStore : JSFile**

Questo attributo rappresenta un riferimento al file CollectionStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-collectionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CollectionActionCreator.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreator.js.

### Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

## DocumentView

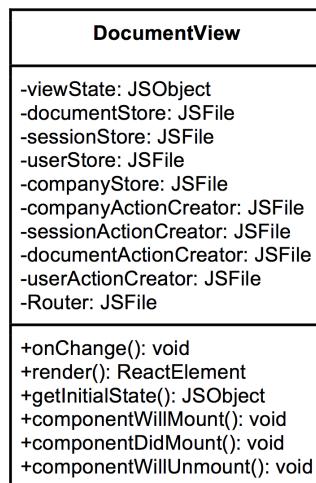


Figura 160: Diagramma della classe DocumentView

## Descrizione

Questa classe descrive la pagina utile a visualizzare il Document selezionata dall'utente attualmente autenticato.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

## Utilizzo

Viene utilizzata per mostrare la pagina di visualizzazione di un documento; quando un utente interagisce con essa, viene chiamato un metodo del DocumentActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte del DocumentStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::DSLISStores::DocumentStore
- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore
- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator
- Front-end::ActionCreators::DSLISActionCreators::DocumentActionCreator

### Attributi

**-viewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-documentStore : JSFile**

Questo attributo rappresenta un riferimento al file DocumentStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-documentActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file DocumentActionCreator.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreator.js.

## Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

## CellView

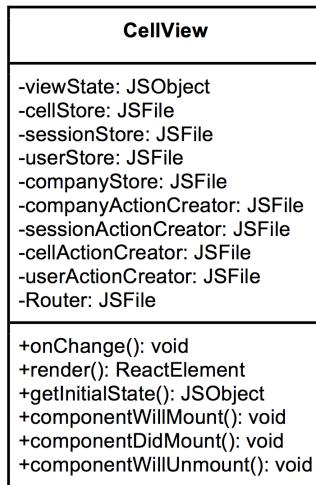


Figura 161: Diagramma della classe CellView

## Descrizione

Questa classe descrive la pagina utile a visualizzare la Cell selezionata dall'utente attualmente autenticato.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

## Utilizzo

Viene utilizzata per mostrare la pagina di visualizzazione di una Cell; quando un utente interagisce con essa, viene chiamato un metodo della CellActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte del CellStore.

## Classi ereditate

- Front-end::Views::ObserverView

## Relazioni con altre classi

- Front-end::Stores::DSLISStores::CellStore
- Front-end::Stores::SessionStore
- Front-end::Stores::UserStore
- Front-end::Stores::CompanyStore
- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator
- Front-end::ActionCreators::DSLISActionCreators::CellActionCreator

## Attributi

### **-ViewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

### **-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

### **-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

### **-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

### **-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-cellStore : JSFile**

Questo attributo rappresenta un riferimento al file CellStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-cellActionCreators : JSFile**

Questo attributo rappresenta un riferimento al file CellActionCreators.js;

**-sessionActionCreators : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreators.js.

## Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

## GestioneUtentiAziendaView

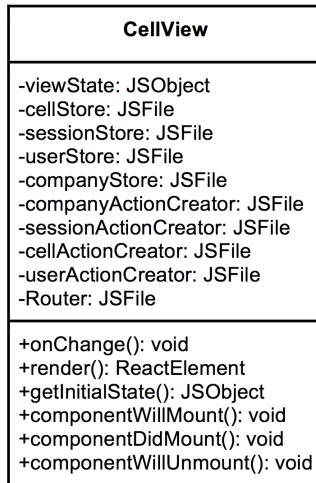


Figura 162: Diagramma della classe CellView

### Descrizione

Questa classe descrive la pagina che consente di visualizzare una serie di opzioni utili all'utente quali:

- Rimuovere un utente dal sistema e in particolare dall'azienda cui fa parte;
- Visualizzare un form nel quale inserire l'email dell'utente da invitare e il ruolo che avrà all'interno dell'azienda;
- Modificare il ruolo di un utente, permettendo la sua selezione e la selezione del tipo di modifica che si vuole attuare, in particolare, promozione o retrocessione.

Rappresenta la componente ConcreteObserver del design pattern Observer.  
Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina di gestione degli utenti appartenenti ad una specifica azienda; quando un utente interagisce con essa, viene chiamato un metodo dello UserActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte dello UserStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::CompanyStore
- Front-end::Stores::UserStore

- Front-end::Stores::SessionStore
- Front-end::Router
- Front-end::ActionCreators::UserActionCreator
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator

### Attributi

**-ViewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-userStore : JSFile**

Questo attributo rappresenta un riferimento al file UserStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-userActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file UserActionCreator.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreator.js.

### Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

### SuperAmministratoreDashboardView

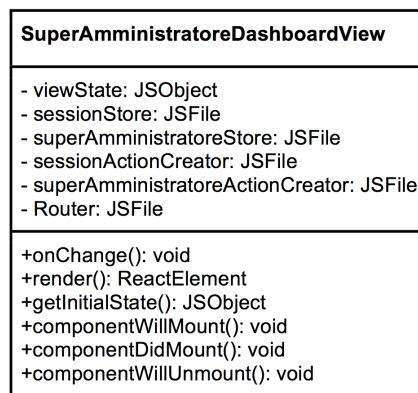


Figura 163: Diagramma della classe SuperAmministratoreDashboardView

### Descrizione

Questa classe descrive la pagina utile a visualizzare la Dashboard, predefinita o personalizzata, propria del profilo del super amministratore.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina nella quale si trova la Dashboard del super amministratore; quando un utente interagisce con essa, viene chiamato un metodo dello DashboardActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte del DashboardStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::SessionStore

- Front-end::Stores::SuperAmministratoreStore
- Front-end::Router
- Front-end::ActionCreators::SuperAmministratoreActionCreator
- Front-end::ActionCreators::SessionActionCreator

### Attributi

**-viewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-superAmministratoreStore : JSFile**

Questo attributo rappresenta un riferimento al file SuperAmministratoreStore.js;

**-superAmministratoreActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SuperAmministratoreActionCreator.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreator.js.

### Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come

ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

### GestioneDatiAziendeView

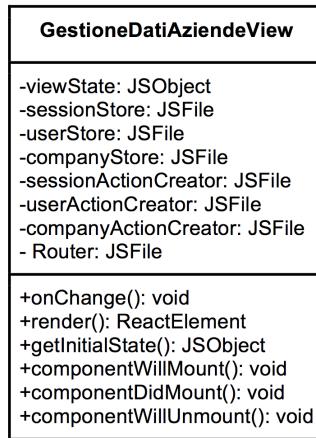


Figura 164: Diagramma della classe GestioneDatiAziendeView

### Descrizione

Questa classe descrive la pagina che consente al super amministratore di gestire i dati relativi ad una specifica azienda registrata nel sistema.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina di gestione dei dati delle aziendali; quando un utente interagisce con essa, viene chiamato un metodo del SuperAmministratoreActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte del CompanyStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::SessionStore
- Front-end::Stores::SuperAmministratoreStore
- Front-end::Stores::CompanyStore

- Front-end::Router
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator
- Front-end::ActionCreators::SuperAmministratoreActionCreator

### Attributi

**-viewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-superAmministratoreStore : JSFile**

Questo attributo rappresenta un riferimento al file SuperAmministratoreStore.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-superAmministratoreActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SuperAmministratoreActionCreator.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreator.js.

### Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

### GestioneDatiUtenteView

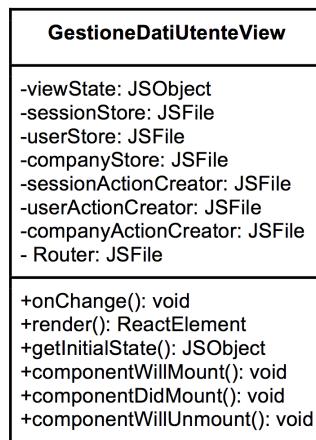


Figura 165: Diagramma della classe GestioneDatiUtenteView

### Descrizione

Questa classe descrive la pagina che consente al super amministratore di gestire i dati relativi a uno specifico utente registrato nel sistema MaaS e di fornire gli strumenti che gli consentono di impersonificarsi in esso.

Rappresenta la componente ConcreteObserver del design pattern Observer.

Viene utilizzato il pattern Module per dare visibilità o meno ai metodi.

### Utilizzo

Viene utilizzata per mostrare la pagina di gestione dei dati di un utente; quando il super amministratore interagisce con essa, viene chiamato un metodo del SuperAmministratoreActionCreator, utile a creare una Action corrispondente all'azione eseguita. Inoltre, riceve i dati utili ad aggiornare il proprio stato da parte del SuperAmministratoreStore.

### Classi ereditate

- Front-end::Views::ObserverView

### Relazioni con altre classi

- Front-end::Stores::SessionStore
- Front-end::Stores::SuperAmministratoreStore
- Front-end::Stores::CompanyStore
- Front-end::Router
- Front-end::ActionCreators::CompanyActionCreator
- Front-end::ActionCreators::SessionActionCreator
- Front-end::ActionCreators::SuperAmministratoreActionCreator

### Attributi

**-viewState : JSObject**

Rappresenta l'oggetto JavaScript contenente le diverse variabili che caratterizzano lo stato della view;

**-Router : JSFile**

Questo attributo rappresenta un riferimento al file Router.js;

**-sessionStore : JSFile**

Questo attributo rappresenta un riferimento al file SessionStore.js;

**-companyStore : JSFile**

Questo attributo rappresenta un riferimento al file CompanyStore.js;

**-superAmministratoreStore : JSFile**

Questo attributo rappresenta un riferimento al file SuperAmministratoreStore.js;

**-companyActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file CompanyActionCreator.js;

**-superAmministratoreActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SuperAmministratoreActionCreator.js;

**-sessionActionCreator : JSFile**

Questo attributo rappresenta un riferimento al file SessionActionCreator.js.

### Metodi

**+onChange() : void**

Questo metodo rappresenta l'implementazione dell'omonimo metodo presente nella classe base ObserverView e si occupa di richiamare il metodo offerto dalle Component API di React `setState(viewState)`;

**+render() : ReactElement**

Questo metodo si occupa di ritornare un singolo elemento figlio che rappresenta una riproduzione virtuale di una componente (React View) DOM nativa, oppure di un'altra componente definita appositamente;

**+getInitialState() : JSObject**

Questo metodo si occupa di ritornare in un oggetto Javascript lo stato iniziale di una view;

**+componentWillMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla composizione della view, appena prima dell'invocazione del metodo `render()`;

**+componentDidMount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla manipolazione del DOM e al recupero dei dati, appena dopo dell'invocazione del metodo `render()`;

**+componentWillUnmount() : void**

Questo metodo si occupa di eseguire determinate azioni utili alla pulizia generale, come ad esempio l'invalidazione dei timer o la pulizia degli elementi DOM che sono stati creati dal metodo `componentDidMount()`. Viene invocato appena prima della deallocazione del componente.

## 7 Design pattern

Un design pattern è una soluzione progettuale elegante e generale ad un problema ricorrente. Si tratta in particolare, di una descrizione o modello logico da applicare per la risoluzione di un problema che può presentarsi in diverse situazioni durante le fasi di progettazione e sviluppo del software, ancor prima della definizione dell'algoritmo risolutivo della parte computazionale. Essi si suddividono in quattro categorie:

- **Architetturali:** esprimono schemi di base per impostare l'organizzazione strutturale di un sistema software;
- **Creazionali:** forniscono un'astrazione del processo di istanziazione degli oggetti;
- **Strutturali:** si occupano delle modalità di composizione di classi e oggetti per formare strutture complesse;
- **Comportamentali:** si occupano di algoritmi e dell'assegnamento di responsabilità tra oggetti collaboranti.

Per un approfondimento dei Design Pattern utilizzati nel progetto si rimanda all'Appendice A; in seguito verranno descritti i Design Pattern implementati.

### 7.1 Design Pattern Architetturali

#### 7.1.1 Flux

##### 7.1.1.1 Scopo

Flux è un pattern architettonico utilizzato per costruire applicazioni web client-server progettato per superare ai limiti imposti dall'MVC. Flux: presenta una flusso circolare, tra Store-View-Action-Dispatcher, di tipo unidirezionale. Questo porta a diversi vantaggi:

- facilità di comprensione del sistema;
- uso di dati esplicativi invece di dati derivati;
- separazione dei dati dallo stato della view;
- evita effetti a cascata e previene aggiornamenti nidificati.

##### 7.1.1.2 Utilizzo

Rappresenta il front-end dell'applicazione e viene utilizzato nell'omonimo package Front-end. Sono presenti le seguenti principali componenti:

- **Dispatcher:** questo componente è unico nell'architettura e costituisce l'hub centrale che gestisce tutto il flusso di dati nell'applicazione. Viene implementato dalla classe Front-end::dispatcher;
- **Stores:** questo componente contiene lo stato logico dell'applicazione e ha il compito di gestire lo stato della View. Viene implementato dal package Front-end::Stores;

- **Views:** rappresenta l'interfaccia grafica dell'applicazione, riceve dati dalle Store ed interazioni dall'utente. Viene implementato dal package Front-end::Views;
- **Actions:** sono oggetti che descrivono le azioni che si vuole compiere, contenenti il nome dell'azione da compiere ed i dati a esse relativi. Essi vengono creati da un componente chiamato ActionCreator che nel nostro sistema viene realizzato nel package Front-end::ActionCreators;
- **WebAPIUtils:** rappresenta l'interfaccia con le API del server e fornisce i metodi utili ad inviare le richieste del client e a ricevere le risposte dal server. Viene implementato dal package Front-end::WebAPIUtils.

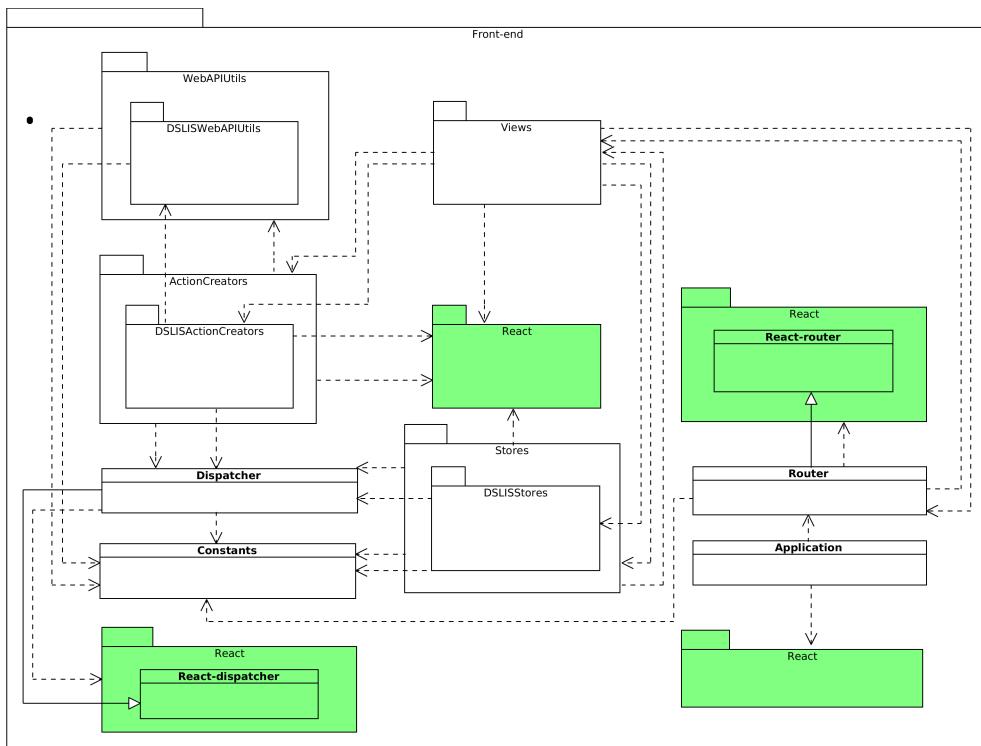


Figura 166: Contestualizzazione del pattern Flux

Il dispatcher, che funge da semaforo, garantisce che ogni callback venga inoltrata agli store solo quando essi sono a disposizione, cioè liberi, e ha la possibilità di cancellare delle action in attesa di esecuzione o di fermarne l'esecuzione, tramite il metodo `waitFor`, fino a quando lo store interessato non ritorna disponibile. Inoltre in caso di dipendenze circolare tra diversi store il dispatcher informa della presenza di questa problematica, che potrà essere risolta per esempio con l'aggiunta di un ulteriore store.

### 7.1.2 Constructor injection

#### 7.1.2.1 Scopo

Constructor injection aiuta la separazione del comportamento di una componente dalla risoluzione delle sue dipendenze

#### 7.1.2.2 Utilizzo

Usato per ridurre l'accoppiamento, in particolare tra le classi CollectionModel, DocumentModel e ConvertTo nel package Back-end::Models::DSLISModels. L'oggetto ConvertTo al posto di essere istanziato dentro ai Model che lo utilizzano, viene passato a loro, mediante il costruttore, per riferimento.

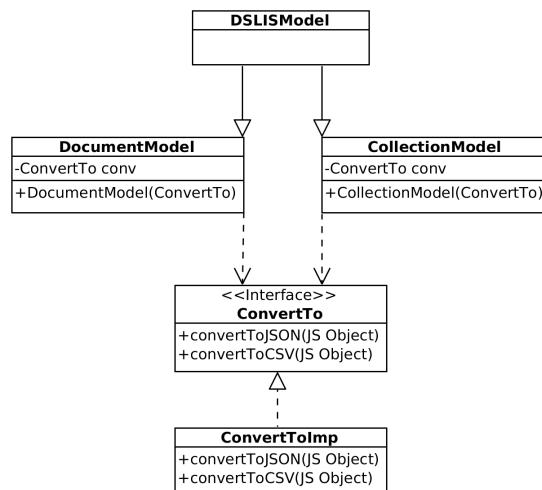


Figura 167: Contestualizzazione del pattern Constructor Injection

## 7.2 Design Pattern Strutturali

### 7.2.1 Module

#### 7.2.1.1 Scopo

Lo scopo è di implementare il concetto di classe convenzionale in JavaScript, usufruendo dei concetti di metodi e variabili pubblici e privati. Viene utilizzato in particolare per definire le funzioni delle classi.

#### 7.2.1.2 Utilizzo

Viene utilizzato nei seguenti package e classi:

- Front-end::Dispatcher;
- Front-end::ActionCreators;
- Front-end::Constants;
- Front-end::Stores;
- Front-end::WebAPIUtils;
- Front-end::Views;
- Front-end::Router;
- Back-end::Models;
- Back-end::Bootstrapper;
- Back-end::Middlewares.

## 7.3 Design Pattern Comportamentali

### 7.3.1 Chain of Responsibility

#### 7.3.1.1 Scopo

Viene utilizzato quando si ha la necessità di disaccoppiare il mittente di una richiesta dal destinatario, nel caso in cui quest'ultimo preveda che le richieste debbano essere gestite da una serie di attori ognuno dei quali con diversa responsabilità e tra loro collegati in modo gerarchico.

#### 7.3.1.2 Utilizzo

Viene utilizzato nel package Back-end::Middlewares ed è composto dai seguenti partecipanti:

- **Handler:** definisce l'interfaccia per gestire le richieste e permette la chiamata a catena ai vari componenti della gerarchia. Viene realizzato dalla seguente classe
  - Back-end::Middlewares::MiddlewareHandler.
- **ConcreteHandler:** gestisce le richieste per le quale è responsabile e se non è in grado di gestire la richiesta, la propaga al suo successore. Viene realizzato dalle seguenti classi:
  - Back-end::Middlewares::UrlNotFoundHandler;
  - Back-end::Middlewares::ServerErrorHandler;
  - Back-end::Middlewares::RequestHandler.
- **Client:** invia una richiesta ad un oggetto concreto, che appartiene alla catena di responsabilità, perchè venga gestita. Viene realizzato dalle seguenti classi:
  - Back-end:: Application.

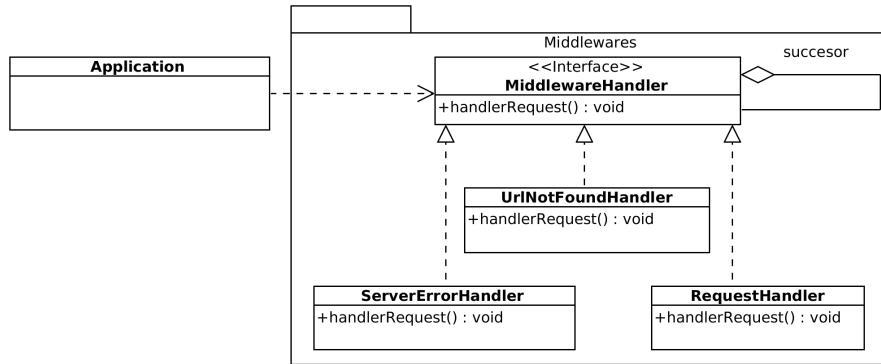


Figura 168: Contestualizzazione del pattern Chain of Responsibility

### 7.3.2 Strategy

#### 7.3.2.1 Scopo

Viene utilizzato per definire una famiglia di algoritmi, incapsularli e renderli intercambiabili. Questo permette di modificare gli algoritmi in modo indipendente dai clienti che fanno uso di essi.

#### 7.3.2.2 Utilizzo

Viene utilizzato nel package Back-end::Models::DSLISModels, in modo da permettere in futuro cambiamenti all'algoritmo di interpretazione del DSL senza dover intervenire sulla classe che ne fa uso. E' composto dai seguenti partecipanti:

- **Strategy:** dichiara una interfaccia che verrà invocata dal Context in base all'algoritmo prescelto. Viene realizzato dalla seguente classe:
  - Back-end::Models::DSLISModels::DSLInterpreterStrategy.
- **ConcreteStrategy:** effettua l'overwrite del metodo del Context al fine di ritornare l'implementazione dell'algoritmo. Viene realizzato dalla seguente classe:
  - Back-end::Models::DSLISModels::DSLInterpreter.
- **Context:** detiene le informazioni di contesto ed ha il compito di invocare l'algoritmo. Viene realizzato dalla seguente classe:
  - Back-end::Models::DSLISModels::DSLISModel.

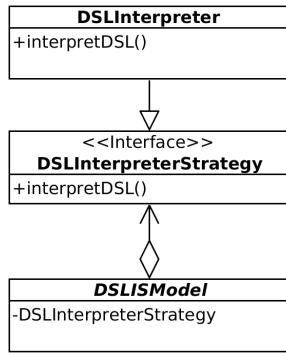


Figura 169: Contestualizzazione del pattern Strategy

### 7.3.3 Observer

#### 7.3.3.1 Scopo

viene utilizzato quando si vuole realizzare una dipendenza uno a molti, in cui il cambiamento di stato di un soggetto (l'osservato), viene notificato a tutte le componenti che si sono registrate ad esso come osservatori.

#### 7.3.3.2 Utilizzo

Viene utilizzato nei package Front-end::Stores e Front-end::Views ed è composto dai seguenti partecipanti:

- **Subject:** espone l'interfaccia che consente agli osservatori di iscriversi e cancellarsi. Viene realizzato dalla seguente classe:
  - Front-end::Stores::SubjectStore;
- **Observer:** espone l'interfaccia che consente di aggiornare gli osservatori in caso di cambio di stato del soggetto osservato. Viene realizzato dalla seguente classe:
  - Front-end::Views::ObserverView;
- **ConcreteSubject:** mantiene lo stato del soggetto osservato e notifica gli osservatori in caso di cambiamento di stato. Viene realizzato dalle seguenti classi:
  - Front-end::Stores::ExternalDatabaseStore;
  - Front-end::Stores::CompanyStore;
  - Front-end::Stores::SuperAmministratoreStore;
  - Front-end::Stores::UserStore;
  - Front-end::Stores::SessionStore;
  - Front-end::Stores::DSLISStores::DocumentStore;
  - Front-end::Stores::DSLISStores::CollectionStore;

- Front-end::Stores::DSLISStores::CellStore;
- Front-end::Stores::DSLISStores::DashboardStore;
- **ConcreteObserver:** implementa l’interfaccia dell’Observer definendo il comportamento in caso di cambio di stato del soggetto osservato. Viene realizzato dalle seguenti classi:
  - Front-end::Views::RegistrazioneView;
  - Front-end::Views::LoginView;
  - Front-end::Views::ModificaPermessiAccessoDatabaseEsterniView;
  - Front-end::Views::RecuperoPasswordView;
  - Front-end::Views::GestioneDSLISView;
  - Front-end::Views::RegistrazioneCollaboratoreView;
  - Front-end::Views::CreazioneDSLISView;
  - Front-end::Views::DashboardView;
  - Front-end::Views::ConfermaRegistrazioneView;
  - Front-end::Views::ImpostazioniApplicazioneView;
  - Front-end::Views::ModificaDashboardPredefinitaView;
  - Front-end::Views::ModificaPasswordView;
  - Front-end::Views::VisualizzazioneDSLISView;
  - Front-end::Views::ModificaDatiAnagraficiView;
  - Front-end::Views::GestioneDatiAziendeView;
  - Front-end::Views::SuperAmministratoreDashboardView;
  - Front-end::Views::CollectionView;
  - Front-end::Views::GestioneDatiUtenteView;
  - Front-end::Views::DocumentView;
  - Front-end::Views::DSLISListView;
  - Front-end::Views::CellView;
  - Front-end::Views::GestioneDatabaseEsterniView;
  - Front-end::Views::GestioneUtentiAziendaView;
  - Front-end::Views::GestionePermessiDSLISView;
  - Front-end::Views::ModificaDSLISView.

## 8 Tracciamento

### 8.1 Tracciamento componenti-requisiti

Componenti	Requisiti
Back-end	
Back-end::Models	RS1O 1 RS1O 1.1 RS1O 1.2 RS1O 1.3 RS1O 1.4 RS1O 14 RS1O 14.1 RS1O 14.1.1 RS1O 14.1.2 RS1O 14.1.3 RS1O 14.1.4

Back-end::Models::DSLISMModels	RS1O 5.7 RS1O 5.7.1 RS1O 5.7.1.1 RS1O 5.7.1.2 RS1O 5.7.1.3 RS1O 5.7.2 RS1O 5.8 RS1O 5.8.1 RS1O 5.8.1.1 RS1O 5.8.1.2 RS1O 5.8.1.3 RS1O 5.8.1.4 RS1O 5.8.2 RS1O 5.8.2.1 RS1O 5.8.2.2 RS1O 5.8.2.3 RS1O 5.8.2.4 RS1O 5.8.2.5 RS1O 5.8.2.6 RS1O 5.8.3 RS1O 5.9 RS1O 5.9.1 RS1O 5.9.2 RS1O 5.9.3 RS1O 5.10 RS1O 5.10.1 RS1O 5.10.2 RS1O 5.10.3 RS1O 5.10.4 RS1O 5.11 RS1O 5.11.1 RS1O 5.11.2 RS1O 5.12 RS1O 5.13
--------------------------------	--

Back-end::Datasources	RS1O 5.15 RS1O 5.16 RS1O 5.17 RS1O 5.18 RS1O 6.1 RS1O 6.1.1 RS1O 6.1.2 RS1O 6.1.3 RS1O 6.2 RS1O 6.3 RS1O 6.3.1 RS1O 6.3.1.1 RS1O 6.3.1.2 RS1O 6.3.2 RS1O 6.3.2.1 RS1O 6.3.2.2 RS1O 8 RS1O 8.2 RS1O 9.2 RS1O 12 RS1O 13 RS1O 13.1 RS1O 13.1.1 RS1O 13.1.4 RS1O 13.1.5 RS1O 13.2 RS1O 13.2.1 RS1O 13.2.3 RS1O 13.2.4 RS1O 13.2.5
Back-end::Connectors	RS1O 5.14.1.2 RS1O 8.2 RS1O 8.3
Back-end::RESTAPIs	
Back-end::RESTAPIs::DSLISRESTAPIs	
Front-end	

Front-end::ActionCreators	RS1O 2.1.2.1 RS1O 2.1.2.2 RS1O 2.1.2.3 RS1O 2.2.1.1 RS1O 2.2.1.2 RS1O 2.2.1.3 RS1O 4.4 RS1O 6.1 RS1O 6.1.1 RS1O 6.1.2 RS1O 6.1.3 RS1O 6.2 RS1O 6.3 RS1O 8.3.1.1 RS1O 8.3.1.2 RS1O 8.3.1.3 RS1O 8.4 RS1O 8.5 RS1O 8.6 RS1O 9.3 RS1O 11 RS1O 13.1.3 RS1O 13.2.2 RS1O 14.4 RS1O 15.3 RS1O 17
Front-end::ActionCreators::DSLISActionCreators	RS1O 5.12
Front-end::WebAPIUtils	RS1O 2.3 RS1O 2.4 RS1O 2.5 RS1O 2.6 RS1O 3.3 RS1O 4 RS1O 4.5
Front-end::WebAPIUtils::DSLISWebAPIUtils	

Front-end::Stores	RS1D 3 RS1D 3.1 RS1O 3.2 RS1D 3.4 RS1D 3.4.1 RS1D 3.4.2 RS1D 3.4.3 RS1D 3.4.4 RS1O 4.3 RS1O 6.3 RS1O 9.1 RS1O 13.1.4 RS1O 13.1.5 RS1O 13.2 RS1O 13.2.3 RS1O 13.2.4 RS1O 13.2.5
Front-end::Stores::DSLISStores	RS1O 5 RS1O 5.3 RS1O 5.6 RS1O 5.6.1 RS1O 5.6.2 RS1O 5.6.3

Front-end::Views	RS1O 1 RS1O 1.1 RS1O 1.2 RS1O 1.3 RS1O 1.4 RS1O 2 RS1O 2.1 RS1O 2.1.1 RS1O 2.1.2 RS1O 2.1.2.1 RS1O 2.1.2.2 RS1O 2.1.2.3 RS1O 2.1.3 RS1O 2.2 RS1O 2.2.1 RS1O 2.2.1.1 RS1O 2.2.1.2 RS1O 2.2.1.3 RS1D 3 RS1D 3.1 RS1O 3.2 RS1D 3.4 RS1D 3.4.1 RS1D 3.4.2 RS1D 3.4.3 RS1D 3.4.4 RS1O 4.1 RS1O 4.2 RS1O 4.3 RS1O 5 RS1F 5.1 RS1O 5.2 RS1O 5.3 RS1O 5.4 RS1O 5.5 RS1O 5.6 RS1O 5.6.1 RS1O 5.6.2 RS1O 5.6.3 RS1O 5.6.4 RS1O 5.6.5 RS1O 5.6.6 RS1O 5.6.7 RS1O 5.14 RS1O 5.14.1 RS1O 5.14.1.1 RS1O 5.14.1.2 RS1O 5.15 RS1O 5.16
------------------	--

Front-end::Views	RS1O 6 RS1O 6.2.1 RS1O 6.2.2 RS1O 6.2.3 RS1O 6.3.1 RS1O 6.3.1.1 RS1O 6.3.1.2 RS1O 6.3.2 RS1O 6.3.2.1 RS1O 6.3.2.2 RS1O 7 RS1O 8.1 RS1O 8.3.1 RS1O 8.3.1.1 RS1O 8.3.1.2 RS1O 8.3.1.3 RS1O 9 RS1O 9.1 RS1O 10 RS1O 12 RS1O 13.1.1 RS1O 13.1.2 RS1O 13.1.4 RS1O 13.1.5 RS1O 13.2.1 RS1O 13.2.3 RS1O 13.2.4 RS1O 13.2.5 RS1O 14.2 RS1O 14.3 RS1O 15 RS1O 15.1 RS1O 15.2 RS1O 16 RS1O 18
------------------	---

Tabella 2: Tracciamento componenti-requisiti

## 8.2 Tracciamento requisiti-componenti

Requisiti	Componenti
-----------	------------

RS1O 1	Back-end::Models Front-end::Views
RS1O 1.1	Back-end::Models Front-end::Views
RS1O 1.2	Back-end::Models Front-end::Views
RS1O 1.3	Back-end::Models Front-end::Views
RS1O 1.4	Back-end::Models Front-end::Views
RS1O 2	Front-end::Views
RS1O 2.1	Front-end::Views
RS1O 2.1.1	Front-end::Views
RS1O 2.1.2	Front-end::Views
RS1O 2.1.2.1	Front-end::ActionCreators Front-end::Views
RS1O 2.1.2.2	Front-end::ActionCreators Front-end::Views
RS1O 2.1.2.3	Front-end::ActionCreators Front-end::Views
RS1O 2.1.3	Front-end::Views
RS1O 2.2	Front-end::Views

RS1O 2.2.1	Front-end::Views
RS1O 2.2.1.1	Front-end::ActionCreators Front-end::Views
RS1O 2.2.1.2	Front-end::ActionCreators Front-end::Views
RS1O 2.2.1.3	Front-end::ActionCreators Front-end::Views
RS1O 2.3	Front-end::WebAPIUtils
RS1O 2.4	Front-end::WebAPIUtils
RS1O 2.5	Front-end::WebAPIUtils
RS1O 2.6	Front-end::WebAPIUtils
RS1D 3	Front-end::Stores Front-end::Views
RS1D 3.1	Front-end::Stores Front-end::Views
RS1O 3.2	Front-end::Stores Front-end::Views
RS1O 3.3	Front-end::WebAPIUtils
RS1D 3.4	Front-end::Stores Front-end::Views
RS1D 3.4.1	Front-end::Stores Front-end::Views

RS1D 3.4.2	Front-end::Stores Front-end::Views
RS1D 3.4.3	Front-end::Stores Front-end::Views
RS1D 3.4.4	Front-end::Stores Front-end::Views
RS1O 4	Front-end::WebAPIUtils
RS1O 4.1	Front-end::Views
RS1O 4.2	Front-end::Views
RS1O 4.3	Front-end::Stores Front-end::Views
RS1O 4.4	Front-end::ActionCreators
RS1O 4.5	Front-end::WebAPIUtils
RS1O 5	Front-end::Stores::DSLISStores Front-end::Views
RS1F 5.1	Front-end::Views
RS1O 5.2	Front-end::Views
RS1O 5.3	Front-end::Stores::DSLISStores Front-end::Views
RS1O 5.4	Front-end::Views

RS1O 5.5	Front-end::Views
RS1O 5.6	Front-end::Stores::DSLISStores Front-end::Views
RS1O 5.6.1	Front-end::Stores::DSLISStores Front-end::Views
RS1O 5.6.2	Front-end::Stores::DSLISStores Front-end::Views
RS1O 5.6.3	Front-end::Stores::DSLISStores Front-end::Views
RS1O 5.6.4	Front-end::Views
RS1O 5.6.5	Front-end::Views
RS1O 5.6.6	Front-end::Views
RS1O 5.6.7	Front-end::Views
RS1O 5.7	Back-end::Models::DSLISModels
RS1O 5.7.1	Back-end::Models::DSLISModels
RS1O 5.7.1.1	Back-end::Models::DSLISModels
RS1O 5.7.1.2	Back-end::Models::DSLISModels

RS1O 5.7.1.3	Back-end::Models::DSLISModels
RS1O 5.7.2	Back-end::Models::DSLISModels
RS1O 5.8	Back-end::Models::DSLISModels
RS1O 5.8.1	Back-end::Models::DSLISModels
RS1O 5.8.1.1	Back-end::Models::DSLISModels
RS1O 5.8.1.2	Back-end::Models::DSLISModels
RS1O 5.8.1.3	Back-end::Models::DSLISModels
RS1O 5.8.1.4	Back-end::Models::DSLISModels
RS1O 5.8.2	Back-end::Models::DSLISModels
RS1O 5.8.2.1	Back-end::Models::DSLISModels
RS1O 5.8.2.2	Back-end::Models::DSLISModels
RS1O 5.8.2.3	Back-end::Models::DSLISModels

RS1O 5.8.2.4	Back-end::Models::DSLISModels
RS1O 5.8.2.5	Back-end::Models::DSLISModels
RS1O 5.8.2.6	Back-end::Models::DSLISModels
RS1O 5.8.3	Back-end::Models::DSLISModels
RS1O 5.9	Back-end::Models::DSLISModels
RS1O 5.9.1	Back-end::Models::DSLISModels
RS1O 5.9.2	Back-end::Models::DSLISModels
RS1O 5.9.3	Back-end::Models::DSLISModels
RS1O 5.10	Back-end::Models::DSLISModels
RS1O 5.10.1	Back-end::Models::DSLISModels
RS1O 5.10.2	Back-end::Models::DSLISModels
RS1O 5.10.3	Back-end::Models::DSLISModels

RS1O 5.10.4	Back-end::Models::DSLISModels
RS1O 5.11	Back-end::Models::DSLISModels
RS1O 5.11.1	Back-end::Models::DSLISModels
RS1O 5.11.2	Back-end::Models::DSLISModels
RS1O 5.12	Back-end::Models::DSLISModels Front-end::ActionCreators::DSLISActionCreators
RS1O 5.13	Back-end::Models::DSLISModels
RS1O 5.14	Front-end::Views
RS1O 5.14.1	Front-end::Views
RS1O 5.14.1.1	Front-end::Views
RS1O 5.14.1.2	Back-end::Connectors Front-end::Views
RS1O 5.15	Back-end::Datasources Front-end::Views
RS1O 5.16	Back-end::Datasources Front-end::Views
RS1O 5.17	Back-end::Datasources

RS1O 5.18	Back-end::Datasources
RS1O 6	Front-end::Views
RS1O 6.1	Back-end::Datasources Front-end::ActionCreators
RS1O 6.1.1	Back-end::Datasources Front-end::ActionCreators
RS1O 6.1.2	Back-end::Datasources Front-end::ActionCreators
RS1O 6.1.3	Back-end::Datasources Front-end::ActionCreators
RS1O 6.2	Back-end::Datasources Front-end::ActionCreators
RS1O 6.2.1	Front-end::Views
RS1O 6.2.2	Front-end::Views
RS1O 6.2.3	Front-end::Views
RS1O 6.3	Back-end::Datasources Front-end::ActionCreators Front-end::Stores
RS1O 6.3.1	Back-end::Datasources Front-end::Views
RS1O 6.3.1.1	Back-end::Datasources Front-end::Views

RS1O 6.3.1.2	Back-end::Datasources Front-end::Views
RS1O 6.3.2	Back-end::Datasources Front-end::Views
RS1O 6.3.2.1	Back-end::Datasources Front-end::Views
RS1O 6.3.2.2	Back-end::Datasources Front-end::Views
RS1O 7	Front-end::Views
RS1O 8	Back-end::Datasources
RS1O 8.1	Front-end::Views
RS1O 8.2	Back-end::Datasources Back-end::Connectors
RS1O 8.3	Back-end::Connectors
RS1O 8.3.1	Front-end::Views
RS1O 8.3.1.1	Front-end::ActionCreators Front-end::Views
RS1O 8.3.1.2	Front-end::ActionCreators Front-end::Views
RS1O 8.3.1.3	Front-end::ActionCreators Front-end::Views
RS1O 8.4	Front-end::ActionCreators

RS1O 8.5	Front-end::ActionCreators
RS1O 8.6	Front-end::ActionCreators
RS1O 9	Front-end::Views
RS1O 9.1	Front-end::Stores Front-end::Views
RS1O 9.2	Back-end::Datasources
RS1O 9.3	Front-end::ActionCreators
RS1O 10	Front-end::Views
RS1O 11	Front-end::ActionCreators
RS1O 12	Back-end::Datasources Front-end::Views
RS1O 13	Back-end::Datasources
RS1O 13.1	Back-end::Datasources
RS1O 13.1.1	Back-end::Datasources Front-end::Views
RS1O 13.1.2	Front-end::Views
RS1O 13.1.3	Front-end::ActionCreators
RS1O 13.1.4	Back-end::Datasources Front-end::Stores Front-end::Views

RS1O 13.1.5	Back-end::Datasources Front-end::Stores Front-end::Views
RS1O 13.2	Back-end::Datasources Front-end::Stores
RS1O 13.2.1	Back-end::Datasources Front-end::Views
RS1O 13.2.2	Front-end::ActionCreators
RS1O 13.2.3	Back-end::Datasources Front-end::Stores Front-end::Views
RS1O 13.2.4	Back-end::Datasources Front-end::Stores Front-end::Views
RS1O 13.2.5	Back-end::Datasources Front-end::Stores Front-end::Views
RS1O 14	Back-end::Models
RS1O 14.1	Back-end::Models
RS1O 14.1.1	Back-end::Models
RS1O 14.1.2	Back-end::Models
RS1O 14.1.3	Back-end::Models
RS1O 14.1.4	Back-end::Models

RS1O 14.2	Front-end::Views
RS1O 14.3	Front-end::Views
RS1O 14.4	Front-end::ActionCreators
RS1O 15	Front-end::Views
RS1O 15.1	Front-end::Views
RS1O 15.2	Front-end::Views
RS1O 15.3	Front-end::ActionCreators
RS1O 16	Front-end::Views
RS1O 17	Front-end::ActionCreators
RS1O 18	Front-end::Views

Tabella 3: Tracciamento requisiti-componenti

### 8.3 Tracciamento classi-requisiti

Classi	Requisiti
Back-end::Bootstrapper	
Back-end::Connectors::EmailConnector	RS1O 5.14.1.2 RS1O 8.2 RS1O 8.3

Back-end::Connectors::MongoDBConnector	RS1O 4 RS1O 4.4
Back-end::Datasources::DatabaseDatasource	RS1O 5.15 RS1O 5.16 RS1O 5.17 RS1O 5.18 RS1O 6.1 RS1O 6.1.1 RS1O 6.1.2 RS1O 6.1.3 RS1O 6.2 RS1O 6.3 RS1O 9.2 RS1O 12 RS1O 13 RS1O 13.1 RS1O 13.2
Back-end::Datasources::DatasourceFacade	RS1O 6.3.1 RS1O 6.3.1.1 RS1O 6.3.1.2 RS1O 6.3.2 RS1O 6.3.2.1 RS1O 6.3.2.2 RS1O 13.1.1 RS1O 13.1.4 RS1O 13.1.5 RS1O 13.2.1 RS1O 13.2.3 RS1O 13.2.4 RS1O 13.2.5
Back-end::Datasources::EmailDatasource	RS1O 8 RS1O 8.2

Back-end::Middlewares::LoopBackErrorHandler	RS1O 1.3 RS1O 2.3 RS1O 2.4 RS1O 2.5 RS1O 3.2 RS1O 3.3 RS1O 4.4 RS1O 5.12 RS1O 5.13 RS1O 8.4 RS1O 8.5 RS1O 9.3 RS1O 11 RS1O 13.1.3 RS1O 13.2.2 RS1O 13.2.6 RS1O 17
Back-end::Middlewares::MiddlewareHandler	
Back-end::Models::CompanyModel	RS1O 13.1.3
Back-end::Models::DSLISModels::CellModel	RS1O 5.7.1.3 RS1O 5.10 RS1O 5.10.1 RS1O 5.10.2 RS1O 5.10.3 RS1O 5.10.4
Back-end::Models::DSLISModels::CollectionAction	RS1O 5.8.3 RS1O 5.11.1 RS1O 5.11.2

Back-end::Models::DSLISMModels::CollectionModel	RS1O 5.7.1.1 RS1O 5.8 RS1O 5.8.1 RS1O 5.8.1.1 RS1O 5.8.1.2 RS1O 5.8.1.3 RS1O 5.8.1.4 RS1O 5.8.2 RS1O 5.8.2.1 RS1O 5.8.2.2 RS1O 5.8.2.3 RS1O 5.8.2.4 RS1O 5.8.2.5 RS1O 5.8.2.6
Back-end::Models::DSLISMModels::DashboardModel	RS1O 5.7 RS1O 5.7.1 RS1O 5.7.2
Back-end::Models::DSLISMModels::DocumentAction	RS1O 5.9.3 RS1O 5.11.1 RS1O 5.11.2
Back-end::Models::DSLISMModels::DocumentModel	RS1O 5.7.1.2 RS1O 5.9 RS1O 5.9.1 RS1O 5.9.2

Back-end::Models::DSLISModels::DSLInterpreter	RS1O 5.7 RS1O 5.7.2 RS1O 5.8 RS1O 5.8.1 RS1O 5.8.1.1 RS1O 5.8.1.2 RS1O 5.8.1.3 RS1O 5.8.1.4 RS1O 5.8.2 RS1O 5.8.2.1 RS1O 5.8.2.2 RS1O 5.8.2.3 RS1O 5.8.2.4 RS1O 5.8.2.5 RS1O 5.8.2.6 RS1O 5.8.3 RS1O 5.9 RS1O 5.9.1 RS1O 5.9.2 RS1O 5.9.3 RS1O 5.10 RS1O 5.10.1 RS1O 5.10.2 RS1O 5.10.3 RS1O 5.10.4 RS1O 5.11 RS1O 5.11.1 RS1O 5.11.2
Back-end::Models::DSLISModels::DSLInterpreterStrategy	
Back-end::Models::DSLISModels::DSLISModel	RS1O 5.11 RS1O 5.12 RS1O 5.13
Back-end::Models::DSLISModels::ExportCollection	RS1O 5.11.1
Back-end::Models::DSLISModels::ExportCSVCollection	RS1O 5.11.1
Back-end::Models::DSLISModels::ExportCSVDocument	RS1O 5.11.1
Back-end::Models::DSLISModels::ExportDocument	RS1O 5.11.1

Back-end::Models::DSLISModels::ExportJSONCollection	RS1O 5.11.1
Back-end::Models::DSLISModels::ExportJSONDocument	RS1O 5.11.1
Back-end::Models::DSLISModels::SendCSVEmailCollection	RS1O 5.11.2
Back-end::Models::DSLISModels::SendCSVEmailDocument	RS1O 5.11.2
Back-end::Models::DSLISModels::SendEmailCollection	RS1O 5.11.2
Back-end::Models::DSLISModels::SendEmailDocument	RS1O 5.11.2
Back-end::Models::DSLISModels::SendJSONEmailCollection	RS1O 5.11.2
Back-end::Models::DSLISModels::SendJSONEmailDocument	RS1O 5.11.2
Back-end::Models::ExternalDatabaseModel	RS1O 4 RS1O 4.4
Back-end::Models::SuperAmministratoreModel	
Back-end::Models::UserModel	RS1D 3 RS1D 3.1 RS1O 3.2 RS1D 3.4 RS1D 3.4.1 RS1D 3.4.2 RS1D 3.4.3 RS1D 3.4.4 RS1O 13.2.2 RS1O 14 RS1O 14.1 RS1O 14.1.1 RS1O 14.1.2 RS1O 14.1.3 RS1O 14.1.4 RS1O 17

LoopBack::LoopBackEmailConnector	RS1O 5.14.1.2 RS1O 8.2 RS1O 8.3
LoopBack::LoopBackErrorHandler	RS1O 1.3 RS1O 2.3 RS1O 2.4 RS1O 2.5 RS1O 3.2 RS1O 3.3 RS1O 4.4 RS1O 5.12 RS1O 5.13 RS1O 8.4 RS1O 8.5 RS1O 9.3 RS1O 11 RS1O 13.1.3 RS1O 13.2.2 RS1O 13.2.6 RS1O 17
LoopBack::LoopBackMongoDBConnector	RS1O 4 RS1O 4.4
LoopBack::LoopBackPersistedModel	RS1O 1 RS1O 1.1 RS1O 1.2 RS1O 1.4
LoopBack::LoopBackurlNotFoundHandler	RS1O 4.4 RS1O 5.12 RS1O 9.3 RS1O 13.1.3
LoopBack::LoopBackUserModel	
Front-end::ActionCreators::CompanyActionCreator	RS1O 13.1.3 RS1O 13.2.2
Front-end::ActionCreators::DSLISActionCreators::CellActionCreator	

Front-end::ActionCreators::DSLISActionCreators::CollectionActionCreator	
Front-end::ActionCreators::DSLISActionCreators::DashboardActionCreator	RS1O 9.2 RS1O 9.3
Front-end::ActionCreators::DSLISActionCreators::DocumentActionCreator	
Front-end::ActionCreators::ExternalDatabaseActionCreator	RS1O 4.4 RS1O 5.12
Front-end::ActionCreators::SessionActionCreator	RS1O 2.1.2.1 RS1O 2.1.2.2 RS1O 2.1.2.3 RS1O 2.2.1.1 RS1O 2.2.1.2 RS1O 2.2.1.3 RS1O 8.3.1 RS1O 8.3.1.1 RS1O 8.3.1.2 RS1O 8.3.1.3 RS1O 8.4 RS1O 8.5 RS1O 8.6 RS1O 15.3
Front-end::ActionCreators::SuperAmministratore	RS1O 14.1 RS1O 14.1.1 RS1O 14.1.2 RS1O 14.1.3 RS1O 14.1.4
Front-end::ActionCreators::UserActionCreator	RS1O 6.1 RS1O 6.1.1 RS1O 6.1.2 RS1O 6.1.3 RS1O 6.2 RS1O 6.3 RS1O 8.3.1 RS1O 11 RS1O 17
Front-end::Dispatcher	

Front-end::Router	
Front-end::Stores::CompanyStore	RS1O 13.1.4 RS1O 13.1.5
Front-end::Stores::DSLISStores::CellStore	RS1O 5 RS1O 5.3 RS1O 5.6 RS1O 5.6.1 RS1O 5.6.2 RS1O 5.6.3
Front-end::Stores::DSLISStores::CollectionStore	RS1O 5 RS1O 5.3 RS1O 5.6 RS1O 5.6.1 RS1O 5.6.2 RS1O 5.6.3
Front-end::Stores::DSLISStores::DashboardStore	RS1O 5 RS1O 5.3 RS1O 5.6 RS1O 5.6.1 RS1O 5.6.2 RS1O 5.6.3 RS1O 9.1
Front-end::Stores::DSLISStores::DocumentStore	RS1O 5 RS1O 5.3 RS1O 5.6 RS1O 5.6.1 RS1O 5.6.2 RS1O 5.6.3
Front-end::Stores::ExternalDatabaseStore	RS1O 4.3
Front-end::Stores::SessionStore	
Front-end::Stores::SubjectStore	
Front-end::Stores::SuperAmministratoreStore	

Front-end::Stores::UserStore	RS1D 3 RS1D 3.1 RS1O 3.2 RS1D 3.4 RS1D 3.4.1 RS1D 3.4.2 RS1D 3.4.3 RS1D 3.4.4 RS1O 6.1 RS1O 6.1.1 RS1O 6.1.2 RS1O 6.1.3 RS1O 6.3 RS1O 13.2 RS1O 13.2.3 RS1O 13.2.4 RS1O 13.2.5
Front-end::Views::CellView	RS1O 7
Front-end::Views::CollectionView	RS1O 7
Front-end::Views::ConfermaRegistrazioneView	
Front-end::Views::CreazioneDSLISView	RS1O 5 RS1O 5.2 RS1O 7
Front-end::Views::DashboardView	RS1O 1.4 RS1O 7 RS1O 9.1
Front-end::Views::DocumentView	RS1O 7
Front-end::Views::DSLISListView	RS1O 5 RS1O 5.6.5 RS1O 7

Front-end::Views::GestioneDatabaseEsterneView	RS1O 4.1 RS1O 4.2 RS1O 4.4 RS1O 7 RS1O 16
Front-end::Views::GestioneDatiAziendeView	RS1O 13.1.1 RS1O 13.1.2 RS1O 13.1.3 RS1O 13.1.4 RS1O 13.1.5 RS1O 16
Front-end::Views::GestioneDatiUtenteView	RS1D 3 RS1D 3.1 RS1O 3.2 RS1O 3.3 RS1D 3.4 RS1D 3.4.1 RS1D 3.4.2 RS1D 3.4.3 RS1D 3.4.4 RS1O 7 RS1O 13.2 RS1O 13.2.3 RS1O 13.2.4 RS1O 16
Front-end::Views::GestioneDSLISView	RS1O 5 RS1F 5.1 RS1O 5.4 RS1O 7 RS1O 5.14 RS1O 5.14.1 RS1O 5.14.1.1 RS1O 5.14.1.2 RS1O 5.15 RS1O 5.16
Front-end::Views::GestionePermessiEsecuzioneDSLISView	RS1O 5

Front-end::Views::GestionePermessiLetturaDSLISView	RS1O 5
Front-end::Views::GestionePermessiScritturaDSLISView	RS1O 5
Front-end::Views::GestionePermessiDSLISView	RS1O 5.6 RS1O 5.6.1 RS1O 5.6.2 RS1O 5.6.3 RS1O 5.6.4 RS1O 5.6.5 RS1O 5.6.6 RS1O 5.6.7 RS1O 7
Front-end::Views::GestioneUtentiAziendaView	RS1O 6 RS1O 6.1 RS1O 6.2 RS1O 6.2.1 RS1O 6.2.2 RS1O 6.2.3 RS1O 6.3.1 RS1O 6.3.1.1 RS1O 6.3.1.2 RS1O 6.3.2 RS1O 6.3.2.1 RS1O 6.3.2.2 RS1O 7 RS1O 12 RS1O 13.2.1 RS1O 13.2.2 RS1O 13.2.5 RS1O 14.2 RS1O 14.3 RS1O 16 RS1O 18
Front-end::Views::ImpostazioniApplicazioneView	RS1O 7 RS1O 9 RS1O 16

Front-end::Views::LoginView	RS1O 1 RS1O 1.1 RS1O 1.2 RS1O 1.3 RS1O 1.4 RS1O 15 RS1O 15.1 RS1O 15.2 RS1O 15.3
Front-end::Views::ModificaDashboardPredefinitaView	RS1O 7 RS1O 9.1 RS1O 10 RS1O 16
Front-end::Views::ModificaDatiAnagraficiView	RS1D 3 RS1D 3.4 RS1D 3.4.1 RS1D 3.4.2 RS1D 3.4.3 RS1D 3.4.4 RS1O 7 RS1O 16
Front-end::Views::ModificaDSLISView	RS1O 5 RS1F 5.1 RS1O 5.3 RS1O 7
Front-end::Views::ModificaPasswordView	RS1O 7 RS1O 16
Front-end::Views::ModificaPermessiAccessoDatabaseEsterniView	RS1O 4.3 RS1O 7 RS1O 8.3.1 RS1O 8.3.1.1 RS1O 8.3.1.2 RS1O 8.3.1.3 RS1O 16
Front-end::Views::ObserverView	

Front-end::Views::RecuperoPasswordView	RS1O 8.1 RS1O 8.3.1 RS1O 8.3.1.1 RS1O 8.3.1.2 RS1O 8.3.1.3
Front-end::Views::RegistrazioneCollaboratoreView	RS1O 2 RS1O 2.2 RS1O 2.2.1 RS1O 2.2.1.1 RS1O 2.2.1.2 RS1O 2.2.1.3 RS1O 2.4 RS1O 2.6
Front-end::Views::RegistrazioneView	RS1O 2 RS1O 2.1 RS1O 2.1.1 RS1O 2.1.2 RS1O 2.1.2.1 RS1O 2.1.2.2 RS1O 2.1.2.3 RS1O 2.1.3 RS1O 2.3 RS1O 2.4 RS1O 2.5 RS1O 2.6
Front-end::Views::SuperAmministratoreDashboardView	RS1O 1.4 RS1O 16
Front-end::Views::VisualizzazioneDSLISView	RS1O 5 RS1O 5.5 RS1O 7
Front-end::WebAPIUtils::CompanyWebAPIUtils	RS1O 2.5
Front-end::WebAPIUtils::DSLISWebAPIUtils::CellWebAPIUtils	
Front-end::WebAPIUtils::DSLISWebAPIUtils::CollectionWebAPIUtils	
Front-end::WebAPIUtils::DSLISWebAPIUtils::DocumentWebAPIUtils	

Front-end::WebAPIUtils::ExternalDatabaseWebAPIUtils	RS1O 4 RS1O 4.4 RS1O 4.5
Front-end::WebAPIUtils::SessionWebAPIUtils	RS1O 2.1.2.1 RS1O 2.1.2.2 RS1O 2.1.2.3 RS1O 2.2.1.1 RS1O 2.2.1.2 RS1O 2.2.1.3 RS1O 2.3 RS1O 2.4 RS1O 2.5 RS1O 2.6 RS1O 3.3
Front-end::WebAPIUtils::UserWebAPIUtils	
React::React-dispatcher	
React::React-router	

Tabella 4: Tracciamento classi-requisiti

## 8.4 Tracciamento requisiti-classi

Requisiti	Classi
RS1O 1	LoopBack::LoopBackPersistedModel Front-end::Views::LoginView
RS1O 1.1	LoopBack::LoopBackPersistedModel Front-end::Views::LoginView
RS1O 1.2	LoopBack::LoopBackPersistedModel Front-end::Views::LoginView

RS1O 1.3	Back-end::Middlewares::LoopBackErrorHandler LoopBack::LoopBackErrorHandler Front-end::Views::LoginView
RS1O 1.4	LoopBack::LoopBackPersistedModel Front-end::Views::DashboardView Front-end::Views::LoginView Front-end::Views::SuperAmministratoreDashboardView
RS1O 2	Front-end::Views::RegistrazioneCollaboratoreView Front-end::Views::RegistrazioneView
RS1O 2.1	Front-end::Views::RegistrazioneView
RS1O 2.1.1	Front-end::Views::RegistrazioneView
RS1O 2.1.2	Front-end::Views::RegistrazioneView
RS1O 2.1.2.1	Front-end::ActionCreators::SessionActionCreator Front-end::Views::RegistrazioneView Front-end::WebAPIUtils::SessionWebAPIUtils
RS1O 2.1.2.2	Front-end::ActionCreators::SessionActionCreator Front-end::Views::RegistrazioneView Front-end::WebAPIUtils::SessionWebAPIUtils
RS1O 2.1.2.3	Front-end::ActionCreators::SessionActionCreator Front-end::Views::RegistrazioneView Front-end::WebAPIUtils::SessionWebAPIUtils
RS1O 2.1.3	Front-end::Views::RegistrazioneView
RS1O 2.2	Front-end::Views::RegistrazioneCollaboratoreView
RS1O 2.2.1	Front-end::Views::RegistrazioneCollaboratoreView

RS1O 2.2.1.1	Front-end::ActionCreators::SessionActionCreator Front-end::Views::RegistrazioneCollaboratoreView Front-end::WebAPIUtils::SessionWebAPIUtils
RS1O 2.2.1.2	Front-end::ActionCreators::SessionActionCreator Front-end::Views::RegistrazioneCollaboratoreView Front-end::WebAPIUtils::SessionWebAPIUtils
RS1O 2.2.1.3	Front-end::ActionCreators::SessionActionCreator Front-end::Views::RegistrazioneCollaboratoreView Front-end::WebAPIUtils::SessionWebAPIUtils
RS1O 2.3	Back-end::Middlewares::LoopBackerrorHandler LoopBack::LoopBackerrorHandler Front-end::Views::RegistrazioneView Front-end::WebAPIUtils::SessionWebAPIUtils
RS1O 2.4	Back-end::Middlewares::LoopBackerrorHandler LoopBack::LoopBackerrorHandler Front-end::Views::RegistrazioneCollaboratoreView Front-end::Views::RegistrazioneView Front-end::WebAPIUtils::SessionWebAPIUtils
RS1O 2.5	Back-end::Middlewares::LoopBackerrorHandler LoopBack::LoopBackerrorHandler Front-end::Views::RegistrazioneView Front-end::WebAPIUtils::CompanyWebAPIUtils Front-end::WebAPIUtils::SessionWebAPIUtils
RS1O 2.6	Front-end::Views::RegistrazioneCollaboratoreView Front-end::Views::RegistrazioneView Front-end::WebAPIUtils::SessionWebAPIUtils
RS1D 3	Back-end::Models::UserModel Front-end::Stores::UserStore Front-end::Views::GestioneDatiUtenteView Front-end::Views::ModificaDatiAnagraficiView

RS1D 3.1	Back-end::Models::UserModel Front-end::Stores::UserStore Front-end::Views::GestioneDatiUtenteView
RS1O 3.2	Back-end::Middlewares::LoopBackErrorHandler Back-end::Models::UserModel LoopBack::LoopBackErrorHandler Front-end::Stores::UserStore Front-end::Views::GestioneDatiUtenteView
RS1O 3.3	Back-end::Middlewares::LoopBackErrorHandler LoopBack::LoopBackErrorHandler Front-end::Views::GestioneDatiUtenteView Front-end::WebAPIUtils::SessionWebAPIUtils
RS1D 3.4	Back-end::Models::UserModel Front-end::Stores::UserStore Front-end::Views::GestioneDatiUtenteView Front-end::Views::ModificaDatiAnagraficiView
RS1D 3.4.1	Back-end::Models::UserModel Front-end::Stores::UserStore Front-end::Views::GestioneDatiUtenteView Front-end::Views::ModificaDatiAnagraficiView
RS1D 3.4.2	Back-end::Models::UserModel Front-end::Stores::UserStore Front-end::Views::GestioneDatiUtenteView Front-end::Views::ModificaDatiAnagraficiView
RS1D 3.4.3	Back-end::Models::UserModel Front-end::Stores::UserStore Front-end::Views::GestioneDatiUtenteView Front-end::Views::ModificaDatiAnagraficiView
RS1D 3.4.4	Back-end::Models::UserModel Front-end::Stores::UserStore Front-end::Views::GestioneDatiUtenteView Front-end::Views::ModificaDatiAnagraficiView

RS1O 4	Back-end::Connectors::MongoDBConnector Back-end::Models::ExternalDatabaseModel LoopBack::LoopBackMongoDBConnector Front-end::WebAPIUtils::ExternalDatabaseWebAPIUtils
RS1O 4.1	Front-end::Views::GestioneDatabaseEsterniView
RS1O 4.2	Front-end::Views::GestioneDatabaseEsterniView
RS1O 4.3	Front-end::Stores::ExternalDatabaseStore Front-end::Views::ModificaPermessiAccessoDatabaseEsterniView
RS1O 4.4	Back-end::Connectors::MongoDBConnector Back-end::Middlewares::LoopBackerrorHandler Back-end::Models::ExternalDatabaseModel LoopBack::LoopBackerrorHandler LoopBack::LoopBackMongoDBConnector LoopBack::LoopBackurlNotFoundHandler Front-end::ActionCreators::ExternalDatabaseActionCreator Front-end::Views::GestioneDatabaseEsterniView Front-end::WebAPIUtils::ExternalDatabaseWebAPIUtils
RS1O 4.5	Front-end::WebAPIUtils::ExternalDatabaseWebAPIUtils
RS1O 5	Front-end::Stores::DSLISStores::CellStore Front-end::Stores::DSLISStores::CollectionStore Front-end::Stores::DSLISStores::DashboardStore Front-end::Stores::DSLISStores::DocumentStore Front-end::Views::CreazioneDSLISView Front-end::Views::DSLISListView Front-end::Views::GestioneDSLISView Front-end::Views::GestionePermessiEsecuzioneDSLISView Front-end::Views::GestionePermessiLetturaDSLISView Front-end::Views::GestionePermessiScritturaDSLISView Front-end::Views::ModificaDSLISView Front-end::Views::VisualizzazioneDSLISView
RS1F 5.1	Front-end::Views::GestioneDSLISView Front-end::Views::ModificaDSLISView

RS1O 5.2	Front-end::Views::CreazioneDSLISView
RS1O 5.3	Front-end::Stores::DSLISStores::CellStore Front-end::Stores::DSLISStores::CollectionStore Front-end::Stores::DSLISStores::DashboardStore Front-end::Stores::DSLISStores::DocumentStore Front-end::Views::ModificaDSLISView
RS1O 5.4	Front-end::Views::GestioneDSLISView
RS1O 5.5	Front-end::Views::VisualizzazioneDSLISView
RS1O 5.6	Front-end::Stores::DSLISStores::CellStore Front-end::Stores::DSLISStores::CollectionStore Front-end::Stores::DSLISStores::DashboardStore Front-end::Stores::DSLISStores::DocumentStore Front-end::Views::GestionePermessiDSLISView
RS1O 5.6.1	Front-end::Stores::DSLISStores::CellStore Front-end::Stores::DSLISStores::CollectionStore Front-end::Stores::DSLISStores::DashboardStore Front-end::Stores::DSLISStores::DocumentStore Front-end::Views::GestionePermessiDSLISView
RS1O 5.6.2	Front-end::Stores::DSLISStores::CellStore Front-end::Stores::DSLISStores::CollectionStore Front-end::Stores::DSLISStores::DashboardStore Front-end::Stores::DSLISStores::DocumentStore Front-end::Views::GestionePermessiDSLISView
RS1O 5.6.3	Front-end::Stores::DSLISStores::CellStore Front-end::Stores::DSLISStores::CollectionStore Front-end::Stores::DSLISStores::DashboardStore Front-end::Stores::DSLISStores::DocumentStore Front-end::Views::GestionePermessiDSLISView
RS1O 5.6.4	Front-end::Views::GestionePermessiDSLISView

RS1O 5.6.5	Front-end::Views::DSLISListView Front-end::Views::GestionePermessiDSLISView
RS1O 5.6.6	Front-end::Views::GestionePermessiDSLISView
RS1O 5.6.7	Front-end::Views::GestionePermessiDSLISView
RS1O 5.7	Back-end::Models::DSLISModels::DashboardModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.7.1	Back-end::Models::DSLISModels::DashboardModel
RS1O 5.7.1.1	Back-end::Models::DSLISModels::CollectionModel
RS1O 5.7.1.2	Back-end::Models::DSLISModels::DocumentModel
RS1O 5.7.1.3	Back-end::Models::DSLISModels::CellModel
RS1O 5.7.2	Back-end::Models::DSLISModels::DashboardModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.8	Back-end::Models::DSLISModels::CollectionModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.8.1	Back-end::Models::DSLISModels::CollectionModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.8.1.1	Back-end::Models::DSLISModels::CollectionModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.8.1.2	Back-end::Models::DSLISModels::CollectionModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.8.1.3	Back-end::Models::DSLISModels::CollectionModel Back-end::Models::DSLISModels::DSLInterpreter

RS1O 5.8.1.4	Back-end::Models::DSLISModels::CollectionModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.8.2	Back-end::Models::DSLISModels::CollectionModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.8.2.1	Back-end::Models::DSLISModels::CollectionModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.8.2.2	Back-end::Models::DSLISModels::CollectionModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.8.2.3	Back-end::Models::DSLISModels::CollectionModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.8.2.4	Back-end::Models::DSLISModels::CollectionModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.8.2.5	Back-end::Models::DSLISModels::CollectionModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.8.2.6	Back-end::Models::DSLISModels::CollectionModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.8.3	Back-end::Models::DSLISModels::CollectionAction Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.9	Back-end::Models::DSLISModels::DocumentModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.9.1	Back-end::Models::DSLISModels::DocumentModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.9.2	Back-end::Models::DSLISModels::DocumentModel Back-end::Models::DSLISModels::DSLInterpreter

RS1O 5.9.3	Back-end::Models::DSLISModels::DocumentAction Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.10	Back-end::Models::DSLISModels::CellModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.10.1	Back-end::Models::DSLISModels::CellModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.10.2	Back-end::Models::DSLISModels::CellModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.10.3	Back-end::Models::DSLISModels::CellModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.10.4	Back-end::Models::DSLISModels::CellModel Back-end::Models::DSLISModels::DSLInterpreter
RS1O 5.11	Back-end::Models::DSLISModels::DSLInterpreter Back-end::Models::DSLISModels::DSLISModel
RS1O 5.11.1	Back-end::Models::DSLISModels::CollectionAction Back-end::Models::DSLISModels::DocumentAction Back-end::Models::DSLISModels::DSLInterpreter Back-end::Models::DSLISModels::ExportCollection Back-end::Models::DSLISModels::ExportCSVCollection Back-end::Models::DSLISModels::ExportCSVDocument Back-end::Models::DSLISModels::ExportDocument Back-end::Models::DSLISModels::ExportJSONCollection Back-end::Models::DSLISModels::ExportJSONDocument

RS1O 5.11.2	Back-end::Models::DSLISModels::CollectionAction Back-end::Models::DSLISModels::DocumentAction Back-end::Models::DSLISModels::DSLInterpreter Back-end::Models::DSLISModels::SendCSVEmailCollection Back-end::Models::DSLISModels::SendCSVEmailDocument Back-end::Models::DSLISModels::SendEmailCollection Back-end::Models::DSLISModels::SendEmailDocument Back-end::Models::DSLISModels::SendJSONEmailCollection Back-end::Models::DSLISModels::SendJSONEmailDocument
RS1O 5.12	Back-end::Middlewares::LoopBackErrorHandler Back-end::Models::DSLISModels::DSLISModel LoopBack::LoopBackErrorHandler LoopBack::LoopBackurlNotFoundHandler Front-end::ActionCreators::ExternalDatabaseActionCreator
RS1O 5.13	Back-end::Middlewares::LoopBackErrorHandler Back-end::Models::DSLISModels::DSLISModel LoopBack::LoopBackErrorHandler
RS1O 5.14	Front-end::Views::GestioneDSLISView
RS1O 5.14.1	Front-end::Views::GestioneDSLISView
RS1O 5.14.1.1	Front-end::Views::GestioneDSLISView
RS1O 5.14.1.2	Back-end::Connectors::EmailConnector LoopBack::LoopBackEmailConnector Front-end::Views::GestioneDSLISView
RS1O 5.15	Back-end::Datasources::DatabaseDatasource Front-end::Views::GestioneDSLISView
RS1O 5.16	Back-end::Datasources::DatabaseDatasource Front-end::Views::GestioneDSLISView
RS1O 5.17	Back-end::Datasources::DatabaseDatasource

RS1O 5.18	Back-end::Datasources::DatabaseDatasource
RS1O 6	Front-end::Views::GestioneUtentiAziendaView
RS1O 6.1	Back-end::Datasources::DatabaseDatasource Front-end::ActionCreators::UserActionCreator Front-end::Stores::UserStore Front-end::Views::GestioneUtentiAziendaView
RS1O 6.1.1	Back-end::Datasources::DatabaseDatasource Front-end::ActionCreators::UserActionCreator Front-end::Stores::UserStore
RS1O 6.1.2	Back-end::Datasources::DatabaseDatasource Front-end::ActionCreators::UserActionCreator Front-end::Stores::UserStore
RS1O 6.1.3	Back-end::Datasources::DatabaseDatasource Front-end::ActionCreators::UserActionCreator Front-end::Stores::UserStore
RS1O 6.2	Back-end::Datasources::DatabaseDatasource Front-end::ActionCreators::UserActionCreator Front-end::Views::GestioneUtentiAziendaView
RS1O 6.2.1	Front-end::Views::GestioneUtentiAziendaView
RS1O 6.2.2	Front-end::Views::GestioneUtentiAziendaView
RS1O 6.2.3	Front-end::Views::GestioneUtentiAziendaView
RS1O 6.3	Back-end::Datasources::DatabaseDatasource Front-end::ActionCreators::UserActionCreator Front-end::Stores::UserStore
RS1O 6.3.1	Back-end::Datasources::DatasourceFacade Front-end::Views::GestioneUtentiAziendaView

RS1O 6.3.1.1	Back-end::Datasources::DatasourceFacade Front-end::Views::GestioneUtentiAziendaView
RS1O 6.3.1.2	Back-end::Datasources::DatasourceFacade Front-end::Views::GestioneUtentiAziendaView
RS1O 6.3.2	Back-end::Datasources::DatasourceFacade Front-end::Views::GestioneUtentiAziendaView
RS1O 6.3.2.1	Back-end::Datasources::DatasourceFacade Front-end::Views::GestioneUtentiAziendaView
RS1O 6.3.2.2	Back-end::Datasources::DatasourceFacade Front-end::Views::GestioneUtentiAziendaView
RS1O 7	Front-end::Views::CellView Front-end::Views::CollectionView Front-end::Views::CreazioneDSLISView Front-end::Views::DashboardView Front-end::Views::DocumentView Front-end::Views::DSLISListView Front-end::Views::GestioneDatabaseEsterniView Front-end::Views::GestioneDatiUtenteView Front-end::Views::GestioneDSLISView Front-end::Views::GestionePermessiDSLISView Front-end::Views::GestioneUtentiAziendaView Front-end::Views::ImpostazioniApplicazioneView Front-end::Views::ModificaDashboardPredefinitaView Front-end::Views::ModificaDatiAnagraficiView Front-end::Views::ModificaDSLISView Front-end::Views::ModificaPasswordView Front-end::Views::ModificaPermessiAccessoDatabaseEsterniView Front-end::Views::VisualizzazioneDSLISView
RS1O 8	Back-end::Datasources::EmailDatasource
RS1O 8.1	Front-end::Views::RecuperoPasswordView

RS1O 8.2	Back-end::Connectors::EmailConnector Back-end::Datasources::EmailDatasource LoopBack::LoopBackEmailConnector
RS1O 8.3	Back-end::Connectors::EmailConnector LoopBack::LoopBackEmailConnector
RS1O 8.3.1	Front-end::ActionCreators::SessionActionCreator Front-end::ActionCreators::UserActionCreator Front-end::Views::ModificaPermessiAccessoDatabaseEsterniView Front-end::Views::RecuperoPasswordView
RS1O 8.3.1.1	Front-end::ActionCreators::SessionActionCreator Front-end::Views::ModificaPermessiAccessoDatabaseEsterniView Front-end::Views::RecuperoPasswordView
RS1O 8.3.1.2	Front-end::ActionCreators::SessionActionCreator Front-end::Views::ModificaPermessiAccessoDatabaseEsterniView Front-end::Views::RecuperoPasswordView
RS1O 8.3.1.3	Front-end::ActionCreators::SessionActionCreator Front-end::Views::ModificaPermessiAccessoDatabaseEsterniView Front-end::Views::RecuperoPasswordView
RS1O 8.4	Back-end::Middlewares::LoopBackErrorHandler LoopBack::LoopBackErrorHandler Front-end::ActionCreators::SessionActionCreator
RS1O 8.5	Back-end::Middlewares::LoopBackErrorHandler LoopBack::LoopBackErrorHandler Front-end::ActionCreators::SessionActionCreator
RS1O 8.6	Front-end::ActionCreators::SessionActionCreator
RS1O 9	Front-end::Views::ImpostazioniApplicazioneView

RS1O 9.1	Front-end::Stores::DSLISStores::DashboardStore Front-end::Views::DashboardView Front-end::Views::ModificaDashboardPredefinitaView
RS1O 9.2	Back-end::Datasources::DatabaseDatasource Front-end::ActionCreators::DSLISActionCreators::DashboardActionCreator
RS1O 9.3	Back-end::Middlewares::LoopBackErrorHandler LoopBack::LoopBackErrorHandler LoopBack::LoopBackurlNotFoundHandler Front-end::ActionCreators::DSLISActionCreators::DashboardActionCreator
RS1O 10	Front-end::Views::ModificaDashboardPredefinitaView
RS1O 11	Back-end::Middlewares::LoopBackErrorHandler LoopBack::LoopBackErrorHandler Front-end::ActionCreators::UserActionCreator
RS1O 12	Back-end::Datasources::DatabaseDatasource Front-end::Views::GestioneUtentiAziendaView
RS1O 13	Back-end::Datasources::DatabaseDatasource
RS1O 13.1	Back-end::Datasources::DatabaseDatasource
RS1O 13.1.1	Back-end::Datasources::DatasourceFacade Front-end::Views::GestioneDatiAziendeView
RS1O 13.1.2	Front-end::Views::GestioneDatiAziendeView
RS1O 13.1.3	Back-end::Middlewares::LoopBackErrorHandler Back-end::Models::CompanyModel LoopBack::LoopBackErrorHandler LoopBack::LoopBackurlNotFoundHandler Front-end::ActionCreators::CompanyActionCreator Front-end::Views::GestioneDatiAziendeView

RS1O 13.1.4	Back-end::Datasources::DatasourceFacade Front-end::Stores::CompanyStore Front-end::Views::GestioneDatiAziendeView
RS1O 13.1.5	Back-end::Datasources::DatasourceFacade Front-end::Stores::CompanyStore Front-end::Views::GestioneDatiAziendeView
RS1O 13.2	Back-end::Datasources::DatabaseDatasource Front-end::Stores::UserStore Front-end::Views::GestioneDatiUtenteView
RS1O 13.2.1	Back-end::Datasources::DatasourceFacade Front-end::Views::GestioneUtentiAziendaView
RS1O 13.2.2	Back-end::Middlewares::LoopBackErrorHandler Back-end::Models::UserModel LoopBack::LoopBackErrorHandler Front-end::ActionCreators::CompanyActionCreator Front-end::Views::GestioneUtentiAziendaView
RS1O 13.2.3	Back-end::Datasources::DatasourceFacade Front-end::Stores::UserStore Front-end::Views::GestioneDatiUtenteView
RS1O 13.2.4	Back-end::Datasources::DatasourceFacade Front-end::Stores::UserStore Front-end::Views::GestioneDatiUtenteView
RS1O 13.2.5	Back-end::Datasources::DatasourceFacade Front-end::Stores::UserStore Front-end::Views::GestioneUtentiAziendaView
RS1O 14	Back-end::Models::UserModel
RS1O 14.1	Back-end::Models::UserModel Front-end::ActionCreators::SuperAmministratore

RS1O 14.1.1	Back-end::Models::UserModel Front-end::ActionCreators::SuperAmministratore
RS1O 14.1.2	Back-end::Models::UserModel Front-end::ActionCreators::SuperAmministratore
RS1O 14.1.3	Back-end::Models::UserModel Front-end::ActionCreators::SuperAmministratore
RS1O 14.1.4	Back-end::Models::UserModel Front-end::ActionCreators::SuperAmministratore
RS1O 14.2	Front-end::Views::GestioneUtentiAziendaView
RS1O 14.3	Front-end::Views::GestioneUtentiAziendaView
RS1O 15	Front-end::Views::LoginView
RS1O 15.1	Front-end::Views::LoginView
RS1O 15.2	Front-end::Views::LoginView
RS1O 15.3	Front-end::ActionCreators::SessionActionCreator Front-end::Views::LoginView
RS1O 16	Front-end::Views::GestioneDatabaseEsterniView Front-end::Views::GestioneDatiAziendeView Front-end::Views::GestioneDatiUtenteView Front-end::Views::GestioneUtentiAziendaView Front-end::Views::ImpostazioniApplicazioneView Front-end::Views::ModificaDashboardPredefinitaView Front-end::Views::ModificaDatiAnagraficiView Front-end::Views::ModificaPasswordView Front-end::Views::ModificaPermessiAccessoDatabaseEsterniView Front-end::Views::SuperAmministratoreDashboardView

RS1O 17	Back-end::Middlewares::LoopBackErrorHandler Back-end::Models::UserModel LoopBack::LoopBackErrorHandler Front-end::ActionCreators::UserActionCreator
RS1O 18	Front-end::Views::GestioneUtentiAziendaView

Tabella 5: Tracciamento requisiti-classi

## 8.5 Tracciamento metodi-test

Metodi	Test
Models::UserModel::createUser()	TU - 1
Models::UserModel::loginUser()	TU - 2
Models::UserModel::editUser()	TU - 3
Models::UserModel::logoutUser()	TU - 4
Models::UserModel::getUsers()	TU - 5
Models::UserModel::getUser()	TU - 6
Models::UserModel::deleteUser()	TU - 7
Models::UserModel::searchUser()	TU - 8
Models::UserModel::forgotPassword()	TU - 9
Models::UserModel::changeRole()	TU - 10
Models::UserModel::sendInvite()	TU - 11
Models::SuperAdminModel::loginSuperAdmin()	TU - 12
Models::SuperAdminModel::logoutSuperAdmin()	TU - 13
Models::SuperAdminModel::impersonateUser()	TU - 14
Models::CompanyModel::getCompanies()	TU - 15
Models::CompanyModel::createCompany()	TU - 16
Models::CompanyModel::getCompany()	TU - 17
Models::CompanyModel::editCompany()	TU - 18
Models::CompanyModel::deleteCompany()	TU - 19
Models::CompanyModel::searchCompany()	TU - 20
Models::ExternalDatabaseModel::addSource()	TU - 21
Models::ExternalDatabaseModel::deleteSource()	TU - 22
Models::ExternalDatabaseModel::getData()	TU - 23
Models::ExternalDatabaseModel::searchDB()	TU - 24
Models::ExternalDatabaseModel::allowAccess()	TU - 25
Models::ExternalDatabaseModel::denyAccess()	TU - 26
Models::DSLISModels::CollectionModel::getCollections()	TU - 27
Models::DSLISModels::CollectionModel::createCollection()	TU - 28
Models::DSLISModels::CollectionModel::editCollection()	TU - 29
Models::DSLISModels::CollectionModel::deleteCollection()	TU - 30
Models::DSLISModels::CollectionModel::retrieveCollection()	TU - 31
Models::DSLISModels::CollectionModel::searchCollection()	TU - 32
Models::DSLISModels::CollectionModel::executeCollection()	TU - 33
Models::DSLISModels::CollectionModel::sendEmail()	TU - 34
Models::DSLISModels::CollectionModel::export()	TU - 35
Models::DSLISModels::CollectionModel::exportDSLIS()	TU - 36
Models::DSLISModels::CollectionModel::importDSLIS()	TU - 37
Models::DSLISModels::DocumentModel::getDocuments()	TU - 38

Models::DSLISModels::DocumentModel::createDocument()	TU - 39
Models::DSLISModels::DocumentModel::editDocument()	TU - 40
Models::DSLISModels::DocumentModel::deleteDocument()	TU - 41
Models::DSLISModels::DocumentModel::searchDocument()	TU - 42
Models::DSLISModels::DocumentModel::executeDocument()	TU - 43
Models::DSLISModels::DocumentModel::executeDocument()	TU - 44
Models::DSLISModels::DocumentModel::sendEmail()	TU - 45
Models::DSLISModels::DocumentModel::export()	TU - 46
Models::DSLISModels::DocumentModel::retrieveDocument()	TU - 47
Models::DSLISModels::DocumentModel::exportDSLIS()	TU - 48
Models::DSLISModels::DocumentModel::importDSLIS()	TU - 49
Models::DSLISModels::DashboardModel::getDashboards()	TU - 50
Models::DSLISModels::DashboardModel::createDashboard()	TU - 51
Models::DSLISModels::DashboardModel::editDashboard()	TU - 52
Models::DSLISModels::DashboardModel::deleteDashboard()	TU - 53
Models::DSLISModels::DashboardModel::retrieveDashboard()	TU - 54
Models::DSLISModels::DashboardModel::searchDashboard()	TU - 55
Models::DSLISModels::DashboardModel::executeDashboard()	TU - 56
Models::DSLISModels::DashboardModel::exportDSLIS()	TU - 57
Models::DSLISModels::DashboardModel::importDSLIS()	TU - 58
Models::DSLISModels::CellModel::getCells()	TU - 59
Models::DSLISModels::CellModel::createCell()	TU - 60
Models::DSLISModels::CellModel::editCell()	TU - 61
Models::DSLISModels::CellModel::deleteCell()	TU - 62
Models::DSLISModels::CellModel::searchCell()	TU - 63
Models::DSLISModels::CellModel::executeCell()	TU - 64
Models::DSLISModels::CellModel::retrieveCell()	TU - 65
Models::DSLISModels::CellModel::exportDSLIS()	TU - 66
Models::DSLISModels::CellModel::importDSLIS()	TU - 67
Models::DSLISModels::ConvertTo::convertToJson()	TU - 68
Models::DSLISModels::ConvertTo::convertToCSV()	TU - 69
Middlewares::MiddlewareHandler::handlerRequest()	TU - 70
Middlewares::urlNotFoundHandler::handlerRequest()	TU - 71
Middlewares::serverErrorHandler::handlerRequest()	TU - 72
Application::init()	TU - 73
Front-end::WebAPIUtils::UserWebAPIUtils::editUserAvatar()	TU - 74
Front-end::WebAPIUtils::UserWebAPIUtils::editUserPersonalData()	TU - 75
Front-end::WebAPIUtils::UserWebAPIUtils::editUserPassword()	TU - 76
Front-end::WebAPIUtils::UserWebAPIUtils::getUsers()	TU - 77
Front-end::WebAPIUtils::UserWebAPIUtils::getUser()	TU - 78
Front-end::WebAPIUtils::UserWebAPIUtils::deleteUser()	TU - 79
Front-end::WebAPIUtils::UserWebAPIUtils::searchUser()	TU - 80
Front-end::WebAPIUtils::UserWebAPIUtils::forgotPassword()	TU - 81
Front-end::WebAPIUtils::UserWebAPIUtils::changeRole()	TU - 82
Front-end::WebAPIUtils::UserWebAPIUtils::sendInvite()	TU - 83
Front-end::WebAPIUtils::CollectionWebAPIUtils::getCollections()	TU - 84
Front-end::WebAPIUtils::CollectionWebAPIUtils::createCollection()	TU - 85
Front-end::WebAPIUtils::CollectionWebAPIUtils::editCollection()	TU - 86
Front-end::WebAPIUtils::CollectionWebAPIUtils::deleteCollection()	TU - 87
Front-end::WebAPIUtils::CollectionWebAPIUtils::retrieveCollectionDSLIS()	TU - 88
Front-end::WebAPIUtils::CollectionWebAPIUtils::searchCollection()	TU - 89
Front-end::WebAPIUtils::CollectionWebAPIUtils::executeCollection()	TU - 90
Front-end::WebAPIUtils::CollectionWebAPIUtils::sendEmail()	TU - 91
Front-end::WebAPIUtils::CollectionWebAPIUtils::export()	TU - 92
Front-end::WebAPIUtils::CollectionWebAPIUtils::exportDSLIS()	TU - 93
Front-end::WebAPIUtils::CollectionWebAPIUtils::importDSLIS()	TU - 94
Front-end::WebAPIUtils::DocumentWebAPIUtils::getDocuments()	TU - 95
Front-end::WebAPIUtils::DocumentWebAPIUtils::createDocument()	TU - 96
Front-end::WebAPIUtils::DocumentWebAPIUtils::editDocument()	TU - 97
Front-end::WebAPIUtils::DocumentWebAPIUtils::deleteDocument()	TU - 98
Front-end::WebAPIUtils::DocumentWebAPIUtils::retrieveDocumentDSLIS()	TU - 99

Front-end::WebAPIUtils::DocumentWebAPIUtils::searchDocument()	TU - 100
Front-end::WebAPIUtils::DocumentWebAPIUtils::executeDocument()	TU - 101
Front-end::WebAPIUtils::DocumentWebAPIUtils::sendEmail()	TU - 102
Front-end::WebAPIUtils::DocumentWebAPIUtils::export()	TU - 103
Front-end::WebAPIUtils::DocumentWebAPIUtils::exportDSLIS()	TU - 104
Front-end::WebAPIUtils::DocumentWebAPIUtils::importDSLIS()	TU - 105
Front-end::WebAPIUtils::DashboardWebAPIUtils::getDashboards()	TU - 106
Front-end::WebAPIUtils::DashboardWebAPIUtils::createDashboard()	TU - 107
Front-end::WebAPIUtils::DashboardWebAPIUtils::editDashboard()	TU - 108
Front-end::WebAPIUtils::DashboardWebAPIUtils::deleteDashboard()	TU - 109
Front-end::WebAPIUtils::DashboardWebAPIUtils::searchDashboard()	TU - 110
Front-end::WebAPIUtils::DashboardWebAPIUtils::executeDashboard()	TU - 111
Front-end::WebAPIUtils::DashboardWebAPIUtils::exportDSLIS()	TU - 112
Front-end::WebAPIUtils::DashboardWebAPIUtils::retrieveDashboardDSLIS()	TU - 113
Front-end::WebAPIUtils::DashboardWebAPIUtils::importDSLIS()	TU - 114
Front-end::WebAPIUtils::CellWebAPIUtils::getCell()	TU - 115
Front-end::WebAPIUtils::CellWebAPIUtils::createCell()	TU - 116
Front-end::WebAPIUtils::CellWebAPIUtils::editCell()	TU - 117
Front-end::WebAPIUtils::CellWebAPIUtils::deleteCell()	TU - 118
Front-end::WebAPIUtils::CellWebAPIUtils::searchCell()	TU - 119
Front-end::WebAPIUtils::CellWebAPIUtils::executeCell()	TU - 120
Front-end::WebAPIUtils::CellWebAPIUtils::exportDSLIS()	TU - 121
Front-end::WebAPIUtils::CellWebAPIUtils::retrieveCellDSLIS()	TU - 122
Front-end::WebAPIUtils::CellWebAPIUtils::importDSLIS()	TU - 123
Front-end::WebAPIUtils::ExternalDatabaseWebAPIUtils::createExternalDatabase()	TU - 124
Front-end::WebAPIUtils::ExternalDatabaseWebAPIUtils::deleteExternalDatabase()	TU - 125
Front-end::WebAPIUtils::ExternalDatabaseWebAPIUtils::getExternalDatabases()	TU - 126
Front-end::WebAPIUtils::ExternalDatabaseWebAPIUtils::searchExternalDatabase()	TU - 127
Front-end::WebAPIUtils::ExternalDatabaseWebAPIUtils::allowExternalDatabaseAccess()	TU - 128
Front-end::WebAPIUtils::ExternalDatabaseWebAPIUtils::denyExternalDatabaseAccess()	TU - 129
Front-end::WebAPIUtils::CompanyWebAPIUtils::getCompanies()	TU - 130
Front-end::WebAPIUtils::CompanyWebAPIUtils::createCompany()	TU - 131
Front-end::WebAPIUtils::CompanyWebAPIUtils::getCompany()	TU - 132
Front-end::WebAPIUtils::CompanyWebAPIUtils::editCompany()	TU - 133
Front-end::WebAPIUtils::CompanyWebAPIUtils::deleteCompany()	TU - 134
Front-end::WebAPIUtils::CompanyWebAPIUtils::searchCompany()	TU - 135
Front-end::WebAPIUtils::SuperAmministratoreWebAPIUtils::impersonateUser()	TU - 136
Front-end::WebAPIUtils::SessionWebAPIUtils::createUser()	TU - 137
Front-end::WebAPIUtils::SessionWebAPIUtils::loginUser()	TU - 138
Front-end::WebAPIUtils::SessionWebAPIUtils::loginSuperAmministratore()	TU - 139
Front-end::WebAPIUtils::SessionWebAPIUtils::logoutUser()	TU - 140
Front-end::WebAPIUtils::SessionWebAPIUtils::logoutSuperAmministratore()	TU - 141
Front-end::ActionCreators::UserActionCreator::response_editUserAvatar()	TU - 142
Front-end::ActionCreators::UserActionCreator::response_editUserPersonalData()	TU - 143
Front-end::ActionCreators::UserActionCreator::response_editUserPassword()	TU - 144
Front-end::ActionCreators::UserActionCreator::response_getUsers()	TU - 145
Front-end::ActionCreators::UserActionCreator::response_getUser()	TU - 146
Front-end::ActionCreators::UserActionCreator::response_deleteUser()	TU - 147
Front-end::ActionCreators::UserActionCreator::response_searchUser()	TU - 148
Front-end::ActionCreators::UserActionCreator::response_forgotPassword():void	TU - 149
Front-end::ActionCreators::UserActionCreator::response_changeRole()	TU - 150
Front-end::ActionCreators::UserActionCreator::response_sendInvite()	TU - 151
Front-end::ActionCreators::CollectionActionCreator::response_getCollections()	TU - 152
Front-end::ActionCreators::CollectionActionCreator::response_createCollection()	TU - 153
Front-end::ActionCreators::CollectionActionCreator::response_editCollection()	TU - 154
Front-end::ActionCreators::CollectionActionCreator::response_deleteCollection()	TU - 155
Front-end::ActionCreators::CollectionActionCreator::response_retrieveCollectionDSLIS()	TU - 156
Front-end::ActionCreators::CollectionActionCreator::response_searchCollection()	TU - 157
Front-end::ActionCreators::CollectionActionCreator::response_executeCollection()	TU - 158
Front-end::ActionCreators::CollectionActionCreator::response_sendEmail()	TU - 159
Front-end::ActionCreators::CollectionActionCreator::response_export()	TU - 160

Front-end::ActionCreators::CollectionActionCreator::response_exportDSLIS()	TU - 161
Front-end::ActionCreators::CollectionActionCreator::response_importDSLIS()	TU - 162
Front-end::ActionCreators::DocumentActionCreator::response_getDocument()	TU - 163
Front-end::ActionCreators::DocumentActionCreator::response_createDocument()	TU - 164
Front-end::ActionCreators::DocumentActionCreator::response_editDocument()	TU - 165
Front-end::ActionCreators::DocumentActionCreator::response_deleteDocument()	TU - 166
Front-end::ActionCreators::DocumentActionCreator::response_searchDocument()	TU - 167
Front-end::ActionCreators::DocumentActionCreator::response_executeDocument()	TU - 168
Front-end::ActionCreators::DocumentActionCreator::response_sendEmail()	TU - 169
Front-end::ActionCreators::DocumentActionCreator::response_exportDSLIS()	TU - 170
Front-end::ActionCreators::DocumentActionCreator::response_retrieveDocumentDSLIS()	TU - 171
Front-end::ActionCreators::DocumentActionCreator::response_export()	TU - 172
Front-end::ActionCreators::DocumentActionCreator::response_importDSLIS()	TU - 173
Front-end::ActionCreators::DashboardActionCreator::response_getDashboards()	TU - 174
Front-end::ActionCreators::DashboardActionCreator::response_createDashboard()	TU - 175
Front-end::ActionCreators::DashboardActionCreator::response_editDashboard()	TU - 176
Front-end::ActionCreators::DashboardActionCreator::response_deleteDashboard()	TU - 177
Front-end::ActionCreators::DashboardActionCreator::response_searchDashboard()	TU - 178
Front-end::ActionCreators::DashboardActionCreator::response_executeDashboard()	TU - 179
Front-end::ActionCreators::DashboardActionCreator::response_exportDSLIS()	TU - 180
Front-end::ActionCreators::DashboardActionCreator::response_retrieveDashboardDSLIS()	TU - 181
Front-end::ActionCreators::DashboardActionCreator::response_importDSLIS()	TU - 182
Front-end::ActionCreators::CellActionCreator::response_getCell()	TU - 183
Front-end::ActionCreators::CellActionCreator::response_createCell()	TU - 184
Front-end::ActionCreators::CellActionCreator::response_editCell()	TU - 185
Front-end::ActionCreators::CellActionCreator::response_deleteCell()	TU - 186
Front-end::ActionCreators::CellActionCreator::response_searchCell()	TU - 187
Front-end::ActionCreators::CellActionCreator::response_executeCell()	TU - 188
Front-end::ActionCreators::CellActionCreator::response_exportDSLIS()	TU - 189
Front-end::ActionCreators::CellActionCreator::response_retrieveCellDSLIS()	TU - 190
Front-end::ActionCreators::CellActionCreator::response_importDSLIS()	TU - 191
Front-end::ActionCreators::ExternalDataBaseActionCreator::response_createExternalDatabase()	TU - 192
Front-end::ActionCreators::ExternalDataBaseActionCreator::response_deleteExternalDatabase()	TU - 193
Front-end::ActionCreators::ExternalDataBaseActionCreator::response_getExternalDatabases()	TU - 194
Front-end::ActionCreators::ExternalDataBaseActionCreator::response_searchExternalDatabase()	TU - 195
Front-end::ActionCreators::ExternalDataBaseActionCreator::response_allowExternalDatabaseAccess()	TU - 196
Front-end::ActionCreators::ExternalDataBaseActionCreator::response_denyExternalDatabaseAccess()	TU - 197
Front-end::ActionCreators::CompanyActionCreator::response_getCompanies()	TU - 198
Front-end::ActionCreators::CompanyActionCreator::response_createCompany()	TU - 199
Front-end::ActionCreators::CompanyActionCreator::response_getCompany()	TU - 200
Front-end::ActionCreators::CompanyActionCreator::response_editCompany()	TU - 201
Front-end::ActionCreators::CompanyActionCreator::response_deleteCompany()	TU - 202
Front-end::ActionCreators::CompanyActionCreator::response_searchCompany()	TU - 203
Front-end::ActionCreators::SessionActionCreator::response_createUser()	TU - 204
Front-end::ActionCreators::SessionActionCreator::response_loginUser()	TU - 205
Front-end::ActionCreators::SessionActionCreator::response_loginSuperAmministratore()	TU - 206
Front-end::ActionCreators::SessionActionCreator::response_logoutUser()	TU - 207
Front-end::ActionCreators::SessionActionCreator::response_logoutSuperAmministratore()	TU - 208
Front-end::Views::ObserverView::onChange()	TU - 209
Front-end::Stores::DSLISstores::DocumentStore::getState()	TU - 210
Front-end::Stores::DSLISstores::DocumentStore::emitChange()	TU - 211
Front-end::Stores::DSLISstores::DocumentStore::addChangeListener()	TU - 212
Front-end::Stores::DSLISstores::DocumentStore::removeChangeListener()	TU - 213
Front-end::Stores::SubjectStore::getState()	TU - 214

Tabella 6: Tracciamento metodi-test

## A Descrizione Design Pattern

### A.1 Design Pattern Architetturali

#### A.1.1 Flux

Flux è un pattern utilizzato per costruire applicazioni web client-server. Flux ha 3 componenti principali:

- **Dispatcher:** questo componente è unico nell'architettura e costituisce l'hub centrale che gestisce tutto il flusso di dati in un'applicazione flux; essenzialmente esso è il componente che si occupa di distribuire le azioni agli store;
- **Stores:** questo componente contiene lo stato logico dell'applicazione e il loro ruolo è quello di gestire lo stato delle componenti della view;
- **Views:** rappresenta l'interfaccia grafica dell'applicazione, riceve dati dallo store e interazioni con l'utente;
- **Actions:** è un oggetto Javascript che descrive l'azione che si vuole compiere;
- **WebAPIUtils:** queste componenti si occupano di interfacciarsi con le API fornite dal server, inviando le richieste del client e ricevendo le risposte dal server.

Le conseguenze derivate dall'utilizzo di questo pattern sono:

- Permette di tracciare con facilità i cambiamenti che avvengono durante lo sviluppo;
- Semplifica individuazione e risoluzione di bug;
- Rende esplicito e facilmente comprensibile il flusso dei dati;
- Aiuta a produrre software scalabile.

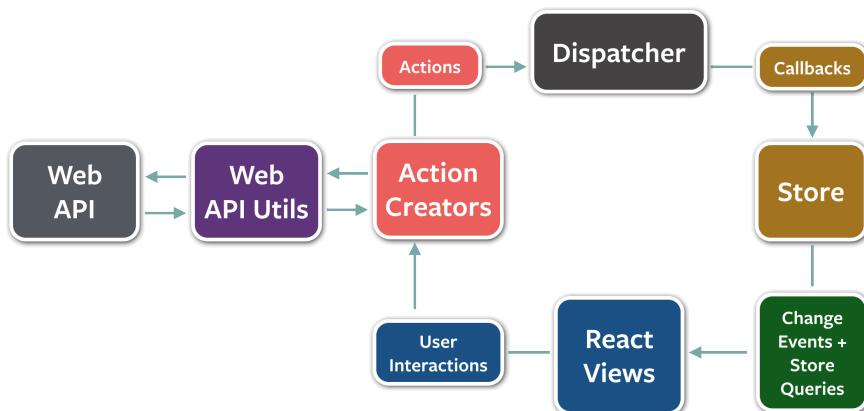


Figura 170: Struttura logica del pattern Flux

### A.1.2 Constructor Injection

Dependency injection è un design pattern il cui scopo è quello di semplificare lo sviluppo e migliorare la testabilità di software di grandi dimensioni, il Constructor Injection è tipo di Dependency Injection.

- l'oggetto servizio da utilizzare;
- l'oggetto client che usufruisce e quindi dipende dal servizio usato;
- l'interfaccia, che definisce come il client può usare il servizio;
- l'injector, che è responsabile per la costruzione del servizio e renderlo disponibile al client.

Le dipendenze, tra servizio e client, vengono dichiarate come parametri del costruttore del client. Il servizio viene quindi passato con il costruttore.

Questo porta a diversi vantaggi:

- Permette di costruire oggetti validi sin dalla loro istanziazione;
- Favorisce la costruzione di oggetti immutabili.

Le conseguenze derivate dall'utilizzo di questo pattern sono:

- Permette di tracciare con facilità i cambiamenti che avvengono durante lo sviluppo;
- Semplifica individuazione e risoluzione di bug;
- Rende esplicito e facilmente comprensibile il flusso dei dati;
- Aiuta a produrre software scalabile.

## A.2 Design Pattern Strutturali

### A.2.1 Module

In JavaScript, il Module pattern è usato per emulare il concetto di classe in cui poter includere sia metodi e variabili pubbliche che private in un singolo oggetto, schermendo determinate parti dallo scope globale. Ciò si traduce in una riduzione della probabilità che i nomi di funzione vadano in conflitto con altre funzioni definite in script aggiuntivi nella pagina. Con questo pattern viene ritornato solo un'API pubblica, mantenendo tutto il resto nello scope privato. L'adozione di questo pattern è una buona scelta per coloro che sono alle prime armi perché è molto più chiaro per gli sviluppatori avere uno sfondo orientato agli oggetti piuttosto dell'idea di vera e propria encapsulazione, almeno nella prospettiva di JavaScript.

### A.3 Design Pattern Comportamentali

#### A.3.1 Observer

Il pattern observer è utile a definire una dipendenza "1...n" fra oggetti, riflettendo la modifica di un oggetto sui dipendenti; il pattern in questione viene applicato per mantenere la consistenza tra gli oggetti e in generale quando ci si trova in uno di questi casi:

- Quando si vuole associare più "viste" differenti ad una astrazione aumentando il grado di riuso dei singoli tipi;
- Quando si hanno diversi oggetti che devono variare nel momento in cui l'oggetto dal quale dipendono modifica il proprio stato e non si
- Nel momento in cui si vuole notificare oggetti senza fare assunzioni su quali siano questi oggetti evitando l'accoppiamento "forte".

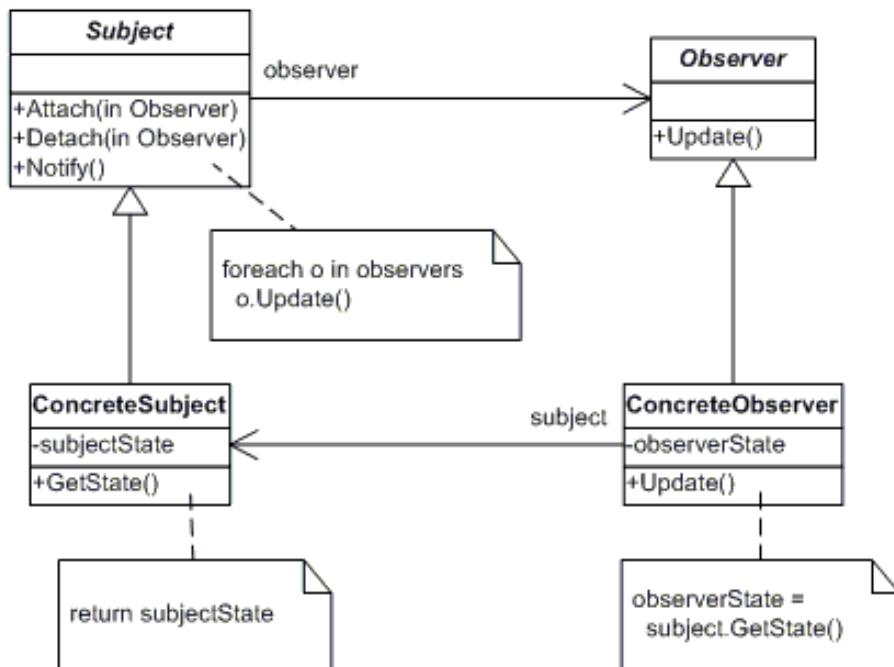


Figura 171: Struttura logica del pattern Observer

#### A.3.2 Strategy

Il pattern Strategy è utile a definire una famiglia di algoritmi, rendendoli interscambiabili, quindi, indipendenti dal client; la sua utilità principale è quella di mantenere la semplicità nel client evitando di inserire tutti gli algoritmi interessati all'interno di esso. Il pattern in questione può venire applicato in diverse circostanze:

- Nel caso in cui diverse classi differiscano solamente per il loro comportamento;

- Nel caso in cui si necessiti di diverse varianti dello stesso algoritmo;
- Nel caso in cui i client debbano utilizzare determinati algoritmi senza interessarsi dei dati che gli algoritmi utilizzano;
- Nel caso in cui una classe definisce differenti comportamenti, tradotti in una serie di statement condizionali.

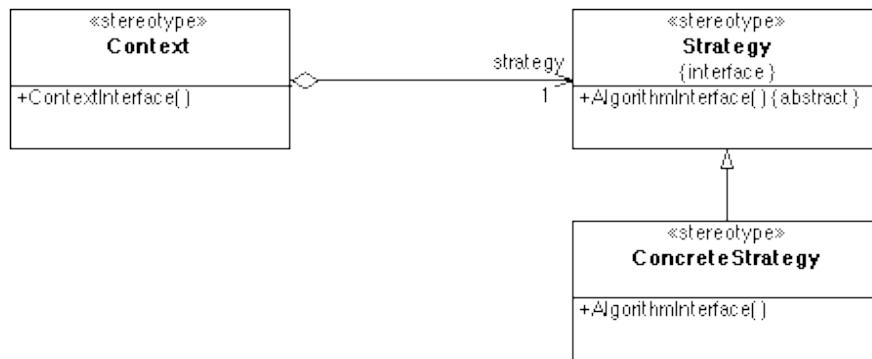


Figura 172: Struttura logica del pattern Strategy

### A.3.3 Chain of Responsibility

Il Chain of Responsibility è un pattern comportamentale che permette di separare i sender delle richieste dai receiver delle stesse. In particolare, la richiesta attraversa una catena di oggetti per poi essere intercettata solo quando raggiunge il proprio gestore. Viene utilizzato in due casi:

- Quando non è possibile determinare staticamente il receiver;
- Quando l'insieme degli oggetti gestori cambia dinamicamente a runtime.

Le richieste vengono dette implicite poiché il sender non ha alcuna conoscenza sull'identità del ricevente; per permettere alla richiesta di attraversare la catena e per rimanere implicita, ogni receiver condivide un'interfaccia comune per gestire le richieste ed accedere al proprio successore. La gerarchia che vorrà inviare richieste dovrà avere una superclasse che dichiara un metodo handler generico. Le conseguenze derivate dall'utilizzo del pattern sono:

- **Riduzione dell'accoppiamento:** gli oggetti non sono a conoscenza di chi gestirà la richiesta ma sanno solo che verrà gestita in modo appropriato; inoltre non bisognerà manutenere i riferimenti a tutti i possibili riceventi;
- **Maggiore flessibilità:** in particolare nell'assegnamento delle responsabilità degli oggetti; è possibile distribuire le responsabilità tra gli oggetti a runtime modificandone la gerarchia mentre asomaticamente è possibile usare il subclassing per specializzare i gestori;
- **Affidabilità:** non c'è garanzia che la request venga gestita, in particolare nel momento in cui la catena non è stata costruita correttamente.

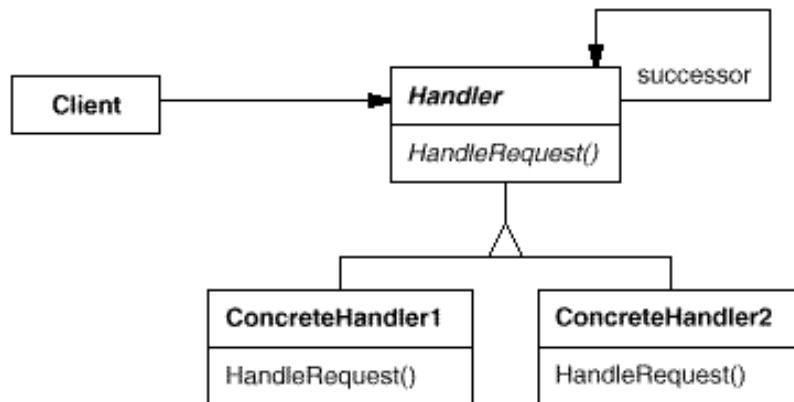


Figura 173: Struttura logica del pattern Chain of Responsibility