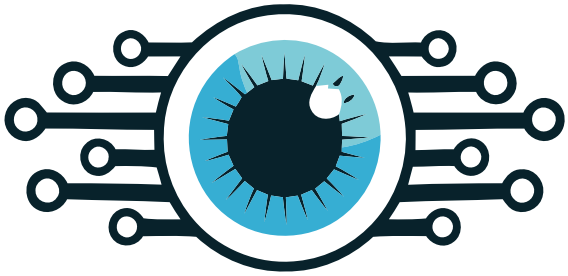
PREMI



Pragma

Definizione di Prodotto

Informazioni sul documento

Versione	3.0.0
Redazione	Giacomo Manzoli
	Massimiliano Baruffato
Verifica	Fabio Vedovato
Approvazione	Daniele Marin
Uso	Esterno
Distribuzione	Pragma
Destinato a	Prof. Vardanega Tullio
	Prof. Cardin Riccardo
	Zucchetti S.p.A.

Sommario

Questo documento descrive la progettazione di dettaglio delle componenti definita dal gruppo Pragma relativa al progetto Premi.

A.A. 2014-15

pragma.swe@gmail.com

Diario delle modifiche

Versione	Descrizione modifica	Autore	Ruolo	Data
3.0.0	*Approvazione del documento*	Daniele Marin	Responsabile	2015-06-14
2.6.1	*Verifica del documento*	Fabio Vedovato	Verificatore	2015-06-12
2.6.0	*Modificata* `Premi::Front-End::Directives::premiPresentation` *a seguito dell'integrazione con ImpressJS*	Giacomo Manzoli	Progettista	2015-06-12
2.5.0	*Rimossa* `Premi::Front-End::Directives::premiPresentationFooter` *e relativo controller*	Giacomo Manzoli	Progettista	2015-06-12
2.4.0	*Adeguata definzione delle classi del package* `Premi::Front-End` *alle nuove norme definite nel documento* Norme di Progetto *v3.1.0*	Giacomo Manzoli	Progettista	2015-06-11
2.3.1	*Verifica del documento*	Fabio Vedovato	Verificatore	2015-06-08
2.3.0	*Aggiunti codici d'errore specifici in appendice A in seguito a E7-2*	Massimiliano Baruffato	Progettista	2015-06-08
2.2.0	*Specificate descrizioni e utilizzo di* `ProjectController`, `UserControllers`, `ErrorHandler` *in seguito agli errori rilevati in sede di RQ*	Massimiliano Baruffato	Progettista	2015-06-08
2.1.0	*Corretto* `presentation` *per le risorse definite in appendice A in seguito agli errori rilevati in sede di RQ*	Massimiliano Baruffato	Progettista	2015-06-08
2.0.0	*Approvazione del documento*	Gianmarco Midena	Responsabile	2015-05-15
1.18.1	*Verifica del documento*	Fabio Vedovato	Verificatore	2015-05-15
1.18.0	*Riviste classi del front-end legate alla presentazione*	Massimiliano Baruffato	Progettista	2015-05-11
1.17.0	*Aggiunte classi* `Front-End::AppRun` *e* `Front-End::AppConfig`	Massimiliano Baruffato	Progettista	2015-05-11
1.16.0	*Rivista organizzazione delle directive e aggiunto un controller esterno per ogni directive*	Giacomo Manzoli	Progettista	2015-05-10
1.15.0	*Aggiunta directive* `premiMindmap`	Giacomo Manzoli	Progettista	2015-05-10

Versione	Descrizione modifica	Autore	Ruolo	Data
1.14.0	*Aggiunto framework Material for Angular*	Giacomo Manzoli	Progettista	2015-05-10
1.13.0	*Aggiunti design pattern Facade B.3 e Observer B.4 all'appendice*	Massimiliano Baruffato	Progettista	2015-05-10
1.12.0	*Modificato metodo dei Routers del BackEnd per uniformarlo alla codifica*	Stefano Munari	Progettista	2015-05-10
1.11.0	*Aggiunta appendice C che descrive il funzionamento di Cytoscape.js*	Giacomo Manzoli	Progettista	2015-05-10
1.10.0	*Aggiornata Struttura, corretta dipendenza su classe astratta*	Stefano Munari	Progettista	2015-05-09
1.9.0	*Aggiornata Gestione generale delle richieste, ora compaiono solo tipi concreti*	Stefano Munari	Progettista	2015-05-09
1.8.0	*Aggiornata Gestione generale delle richieste, ora compaiono solo tipi concreti*	Stefano Munari	Progettista	2015-05-09
1.7.0	*Aggiunta convenzione sui tipi di ritorno per i metodi del Back-End*	Stefano Munari	Progettista	2015-05-09
1.6.0	*Rimosso* `Front-End::Controllers::LogoutController` *e relativa view per decisione E5-2*	Massimiliano Baruffato	Progettista	2015-05-08
1.5.0	*Aggiunta classe* `Premi::Front-End::Model::ErrorInfo`	Giacomo Manzoli	Progettista	2015-05-08
1.4.0	*Riorganizzato package* `Premi::Front-End::Services`	Giacomo Manzoli	Progettista	2015-05-08
1.3.0	*Rinominata classe* `PathStep` *in* `NodeReference`	Massimiliano Baruffato	Progettista	2015-05-08
1.2.0	*Aggiornate le classi* `Premi::Front-End::Model::Node` *e* `Premi::Front-End::Model::NodeContent`	Giacomo Manzoli	Progettista	2015-05-08
1.1.0	*Spostata sezione relativa alle norme progettuali nel documento* Norme di Progetto *v2.1.0*	Giacomo Manzoli	Progettista	2015-05-08
1.0.0	*Approvazione del documento*	Massimiliano Baruffato	Responsabile	2015-05-19
0.44.1	*Verifica delle componenti del front end*	Stefano Munari	Verificatore	2015-04-18
0.44.1	*Verifica della sezione §1*	Stefano Munari	Verificatore	2015-04-18

Versione	Descrizione modifica	Autore	Ruolo	Data
0.44.1	*Verifica delle componenti back end*	Giacomo Manzoli	Verificatore	2015-04-17
0.44.1	*Verifica dell'appendice A e della sezione relativa alle norme progettuali*	Giacomo Manzoli	Verificatore	2015-04-17
0.44.0	*Inseriti riferimenti informativi mancanti*	Stefano Munari	Progettista	2015-04-16
0.43.0	*Completata l'appendice Interfaccia REST*	Andrea Ongaro	Progettista	2015-04-16
0.42.0	*Stesura diagramma di sequenza per Back-End §4.2.3.11*	Stefano Munari	Progettista	2015-04-15
0.41.0	*Stesura diagramma di sequenza per Back-End §4.2.3.10*	Stefano Munari	Progettista	2015-04-15
0.40.0	*Stesura diagramma di sequenza per Back-End §4.2.3.9*	Stefano Munari	Progettista	2015-04-15
0.39.0	*Stesura diagramma di sequenza per Back-End §4.2.3.8*	Stefano Munari	Progettista	2015-04-15
0.38.0	*Stesura diagramma di sequenza per Back-End §4.2.3.7*	Stefano Munari	Progettista	2015-04-15
0.37.0	*Stesura diagramma di sequenza per Back-End §4.2.3.6*	Stefano Munari	Progettista	2015-04-15
0.36.0	*Stesura diagramma di sequenza per Back-End §4.2.3.5*	Stefano Munari	Progettista	2015-04-15
0.35.0	*Stesura diagramma di sequenza per Back-End §4.2.3.4*	Stefano Munari	Progettista	2015-04-15
0.34.0	*Stesura diagramma di sequenza per Back-End §4.2.3.3*	Stefano Munari	Progettista	2015-04-15
0.33.0	*Stesura diagramma di sequenza per Back-End §4.2.3.2*	Stefano Munari	Progettista	2015-04-15
0.32.0	*Stesura diagramma di sequenza per Back-End §4.2.3.1*	Stefano Munari	Progettista	2015-05-15
0.31.0	*Stesura diagramma di sequenza per Back-End §4.2.2.3*	Stefano Munari	Progettista	2015-04-15
0.30.0	*Stesura diagramma di sequenza per Back-End §4.2.2.2*	Stefano Munari	Progettista	2015-04-15
0.29.0	*Stesura diagramma di sequenza per Back-End §4.2.2.1*	Stefano Munari	Progettista	2015-04-15

Versione	Descrizione modifica	Autore	Ruolo	Data
0.28.0	*Stesura diagramma di sequenza per Back-End §4.2.1*	Stefano Munari	Progettista	2015-04-15
0.27.0	*Stesura diagrammi di sequenza relativi al Front-end §4.1.1.5*	Fabio Vedovato	Progettista	2015-04-14
0.26.0	*Stesura diagrammi di sequenza relativi al Front-end §4.1.1.4*	Fabio Vedovato	Progettista	2015-04-14
0.25.0	*Stesura diagrammi di sequenza relativi al Front-end §4.1.1.3*	Fabio Vedovato	Progettista	2015-04-14
0.24.0	*Stesura diagrammi di sequenza relativi al Front-end §4.1.1.2*	Fabio Vedovato	Progettista	2015-04-14
0.23.0	*Stesura diagrammi di sequenza relativi al Front-end §4.1.1*	Fabio Vedovato	Progettista	2015-04-14
0.22.0	*Stesura della sezione Premi::Front-End::Models*	Gianmarco Midena	Progettista	2015-04-13
0.21.0	*Stesura della sezione Premi::Back-End::App::Models*	Andrea Ongaro	Progettista	2015-04-13
0.20.0	*Stesura della sezione Premi::Front-End::Directives*	Daniele Marin	Progettista	2015-04-12
0.19.0	*Stesura della sezione Premi::Front-End::Services*	Gianmarco Midena	Progettista	2015-04-11
0.18.0	*Stesura della sezione Premi::Front-End::Controllers*	Fabio Vedovato	Progettista	2015-04-10
0.17.0	*Stesura della sezione Premi::Front-End::Views*	Fabio Vedovato	Progettista	2015-04-10
0.16.0	*Stesura della sezione Premi::Front-End*	Daniele Marin	Progettista	2015-04-09
0.15.0	*Stesura della sezione Premi::Back-End::App::Controllers::Errors*	Stefano Munari	Progettista	2015-04-09
0.14.0	*Stesura della sezione Premi::Back-End::App::Controllers::Users*	Stefano Munari	Progettista	2015-04-09
0.13.0	*Stesura della sezione Premi::Back-End::App::Controllers:: Projects*	Andrea Ongaro	Progettista	2015-04-09
0.12.0	*Stesura della sezione Premi::Front-End::Directives*	Fabio Vedovato	Progettista	2015-04-08
0.11.0	*Stesura della sezione Premi::Back-End::App::Controllers*	Andrea Ongaro	Progettista	2015-04-07

Versione	Descrizione modifica	Autore	Ruolo	Data
0.10.0	*Stesura della sezione Premi::Back-End::App::Routers*	Stefano Munari	Progettista	2015-04-06
0.9.0	*Stesura della sezione Premi::Back-End::App::Views*	Andrea Ongaro	Progettista	2015-04-05
0.8.0	*Stesura della sezione Premi::Back-End::Config*	Andrea Ongaro	Progettista	2015-04-05
0.7.0	*Stesura della sezione Premi::Back-End e Premi::Back-End::App*	Andrea Ongaro	Progettista	2015-04-05
0.6.0	*Inizio stesura della sezione Specifica del Back-End*	Andrea Ongaro	Progettista	2015-04-05
0.5.0	*Inserito pattern Dependecy Injection in appendice*	Gianmarco Midena	Progettista	2015-04-04
0.4.0	*Stesura sezione relativa alle norme progettuali*	Andrea Ongaro	Progettista	2015-04-03
0.3.0	*Inseriti riferimenti informativi relativi ai middleware Express*	Stefano Munari	Progettista	2015-04-02
0.2.0	*Inserito pattern Chain of Responsibility in appendice*	Stefano Munari	Progettista	2015-04-02
0.1.0	*Inizio stesura dell'appendice Interfaccia REST*	Andrea Ongaro	Progettista	2015-04-02
0.0.0	*Creazione del documento*	Gianmarco Midena	Progettista	2015-31-03

Tabella 1: Diario delle modifiche.

Indice

1 Introduzione **14**
1.1 Scopo del documento . . . 14
1.2 Scopo del prodotto . . . 14
1.3 Glossario . . . 14
1.4 Riferimenti . . . 14
1.4.1 Normativi . . . 14
1.4.2 Informativi . . . 14

2 Specifica del Front-End **16**
2.1 Premi::Front-End . . . 16
2.1.1 Informazioni generali . . . 16
2.1.2 Classi . . . 17
2.1.2.1 Premi::Front-End::AppConfig . . . 17
2.1.2.2 Premi::Front-End::AppRouter . . . 17
2.1.2.3 Premi::Front-End::AppRun . . . 18
2.2 Premi::Front-End::Controllers . . . 19
2.2.1 Informazioni generali . . . 19
2.2.2 Classi . . . 20
2.2.2.1 Premi::Front-End::Controllers::AddToPathController . . . 20
2.2.2.2 Premi::Front-End::Controllers::AssociationAdderController . . . 22
2.2.2.3 Premi::Front-End::Controllers::ContextMenuController . . . 23
2.2.2.4 Premi::Front-End::Controllers::DashboardController . . . 24
2.2.2.5 Premi::Front-End::Controllers::EditableNodeContentController . . . 26
2.2.2.6 Premi::Front-End::Controllers::ErrorMessageController . . . 27
2.2.2.7 Premi::Front-End::Controllers::HeaderController . . . 28
2.2.2.8 Premi::Front-End::Controllers::HierarchicalMenuController . . . 30
2.2.2.9 Premi::Front-End::Controllers::LoginController . . . 31
2.2.2.10 Premi::Front-End::Controllers::MindmapEditorController . . . 33
2.2.2.11 Premi::Front-End::Controllers::NodeContentsEditorController . . . 36
2.2.2.12 Premi::Front-End::Controllers::NodeController . . . 38
2.2.2.13 Premi::Front-End::Controllers::PathController . . . 40
2.2.2.14 Premi::Front-End::Controllers::PathsEditorController . . . 41
2.2.2.15 Premi::Front-End::Controllers::PathsListController . . . 43
2.2.2.16 Premi::Front-End::Controllers::PresentationController . . . 45
2.2.2.17 Premi::Front-End::Controllers::PresentationViewerController . . . 47
2.2.2.18 Premi::Front-End::Controllers::ProjectSettingsEditorController . . . 50
2.2.2.19 Premi::Front-End::Controllers::ProjectsListController . . . 52
2.2.2.20 Premi::Front-End::Controllers::RegistrationController . . . 53
2.2.2.21 Premi::Front-End::Controllers::SmartMenuController . . . 55
2.3 Premi::Front-End::Directives . . . 56
2.3.1 Informazioni generali . . . 56
2.3.2 Classi . . . 56
2.3.2.1 Premi::Front-End::Directives::premiAddToPath . . . 56
2.3.2.2 Premi::Front-End::Directives::premiAssociationAdder . . . 57
2.3.2.3 Premi::Front-End::Directives::premiContextMenu . . . 58
2.3.2.4 Premi::Front-End::Directives::premiEditableNodeContent . . . 60
2.3.2.5 Premi::Front-End::Directives::premiErrorMessage . . . 61
2.3.2.6 Premi::Front-End::Directives::premiHeader . . . 62
2.3.2.7 Premi::Front-End::Directives::premiHierarchicalMenu . . . 63
2.3.2.8 Premi::Front-End::Directives::premiMindmap . . . 64
2.3.2.9 Premi::Front-End::Directives::premiNode . . . 65

2.3.2.10 Premi::Front-End::Directives::premiNodeContent 66
2.3.2.11 Premi::Front-End::Directives::premiNodeContentsEditor 67
2.3.2.12 Premi::Front-End::Directives::premiPath 69
2.3.2.13 Premi::Front-End::Directives::premiPathsList 70
2.3.2.14 Premi::Front-End::Directives::premiPresentation 71
2.3.2.15 Premi::Front-End::Directives::premiProjectSettingsEditor 72
2.3.2.16 Premi::Front-End::Directives::premiProjectsList 73
2.3.2.17 Premi::Front-End::Directives::premiSmartMenu 75
2.4 Premi::Front-End::Model 76
2.4.1 Informazioni generali 76
2.4.2 Classi 76
2.4.2.1 Premi::Front-End::Model::ErrorInfo 76
2.4.2.2 Premi::Front-End::Model::Node 78
2.4.2.3 Premi::Front-End::Model::NodeContent 80
2.4.2.4 Premi::Front-End::Model::NodeReference 83
2.4.2.5 Premi::Front-End::Model::Path 84
2.4.2.6 Premi::Front-End::Model::PresentationNode 87
2.4.2.7 Premi::Front-End::Model::Project 88
2.5 Premi::Front-End::Services 91
2.5.1 Informazioni generali 91
2.5.2 Classi 92
2.5.2.1 Premi::Front-End::Services::AuthenticationService 92
2.5.2.2 Premi::Front-End::Services::MindmapAdapterService 94
2.5.2.3 Premi::Front-End::Services::MindmapService 98
2.5.2.4 Premi::Front-End::Services::PathService 102
2.5.2.5 Premi::Front-End::Services::PresentationService 105
2.5.2.6 Premi::Front-End::Services::ProjectService 108
2.5.2.7 Premi::Front-End::Services::ServerURL 111
2.6 Premi::Front-End::Views 112
2.6.1 Informazioni generali 112
2.6.2 Classi 112
2.6.2.1 Premi::Front-End::Views::DashboardView 112
2.6.2.2 Premi::Front-End::Views::LoginView 113
2.6.2.3 Premi::Front-End::Views::MindmapEditorView 113
2.6.2.4 Premi::Front-End::Views::PathsEditorView 115
2.6.2.5 Premi::Front-End::Views::PresentationView 116
2.6.2.6 Premi::Front-End::Views::RegistrationView 117

3 Specifica del Back-End 119
3.1 Premi::Back-End 119
3.1.1 Informazioni generali 119
3.1.2 Classi 119
3.1.2.1 Premi::Back-End::Server 119
3.2 Premi::Back-End::App 120
3.2.1 Informazioni generali 120
3.3 Premi::Back-End::App::Controllers 121
3.3.1 Informazioni generali 121
3.3.2 Classi 122
3.3.2.1 Premi::Back-End::App::Controllers::ErrorHandler 122
3.3.2.2 Premi::Back-End::App::Controllers::NotFoundHandler 123
3.3.2.3 Premi::Back-End::App::Controllers::ProjectController 124
3.3.2.4 Premi::Back-End::App::Controllers::StaticController 125
3.3.2.5 Premi::Back-End::App::Controllers::UserController 126
3.4 Premi::Back-End::App::Controllers::Errors 126

3.4.1 Informazioni generali . . . 126
3.4.2 Classi . . . 126
3.4.2.1 Premi::Back-End::App::Controllers::Errors::PremiError . . . 126
3.5 Premi::Back-End::App::Controllers::Projects . . . 127
3.5.1 Informazioni generali . . . 127
3.5.2 Classi . . . 128
3.5.2.1 Premi::Back-End::App::Controllers::Projects::NodeController . . . 128
3.5.2.2 Premi::Back-End::App::Controllers::Projects::PathController . . . 131
3.5.2.3 Premi::Back-End::App::Controllers::Projects::ProjectManagementController 134
3.6 Premi::Back-End::App::Controllers::Users . . . 137
3.6.1 Informazioni generali . . . 137
3.6.2 Classi . . . 137
3.6.2.1 Premi::Back-End::App::Controllers::Users::AuthenticationController . 137
3.6.2.2 Premi::Back-End::App::Controllers::Users::AuthorizationController 139
3.7 Premi::Back-End::App::Models . . . 139
3.7.1 Informazioni generali . . . 139
3.7.2 Classi . . . 140
3.7.2.1 Premi::Back-End::App::Models::NodeContentModel . . . 140
3.7.2.2 Premi::Back-End::App::Models::NodeModel . . . 141
3.7.2.3 Premi::Back-End::App::Models::PathModel . . . 142
3.7.2.4 Premi::Back-End::App::Models::ProjectModel . . . 144
3.7.2.5 Premi::Back-End::App::Models::RelationModel . . . 150
3.7.2.6 Premi::Back-End::App::Models::Session . . . 151
3.7.2.7 Premi::Back-End::App::Models::UserModel . . . 152
3.8 Premi::Back-End::App::Routers . . . 153
3.8.1 Informazioni generali . . . 153
3.8.2 Classi . . . 154
3.8.2.1 Premi::Back-End::App::Routers::ProjectRouter . . . 154
3.8.2.2 Premi::Back-End::App::Routers::StaticRouter . . . 155
3.8.2.3 Premi::Back-End::App::Routers::UserRouter . . . 155
3.9 Premi::Back-End::App::Views . . . 156
3.9.1 Informazioni generali . . . 156
3.9.2 Classi . . . 157
3.9.2.1 Premi::Back-End::App::Views::UserManualView . . . 157
3.10 Premi::Back-End::Config . . . 157
3.10.1 Informazioni generali . . . 157
3.10.2 Classi . . . 158
3.10.2.1 Premi::Back-End::Config::Config . . . 158

4 Diagrammi di sequenza 159
4.1 Front-End . . . 159
4.1.1 Controllers . . . 159
4.1.1.1 RegistrationController - Registrazione di un nuovo utente . . . 159
4.1.1.2 DashboardController - Apertura di un progetto . . . 160
4.1.1.3 MindmapEditorController - Costruzione del controller . . . 160
4.1.1.4 MindmapEditorController - Selezione di un nodo . . . 161
4.1.1.5 PresentationController - Creazione di una presentazione . . . 162
4.2 Back-End . . . 164
4.2.1 Gestione generale delle richieste . . . 164
4.2.2 Gestione dei fallimenti . . . 165
4.2.2.1 Fallimento della procedura di registrazione . . . 165
4.2.2.2 Fallimento del vincolo utente autenticato . . . 166
4.2.2.3 Fallimento della richiesta di una lista di progetti . . . 166
4.2.3 Richieste REST . . . 167

4.2.3.1 DELETE /profile . . . 167
4.2.3.2 DELETE /projects/:projectId . . . 168
4.2.3.3 PUT /projects/:projectId . . . 168
4.2.3.4 GET /projects/:projectId . . . 169
4.2.3.5 PUT /projects/:projectId/nodes/:nodeId . . . 169
4.2.3.6 POST /projects/:projectId/associations . . . 170
4.2.3.7 DELETE /projects/:projectId/associations/:associationId . . . 171
4.2.3.8 PUT /projects/:projectId/path/:pathId . . . 171
4.2.3.9 POST /projects/:projectId/path/:pathId . . . 172
4.2.3.10 GET /projects/:projectId/premiPresentation . . . 172
4.2.3.11 GET /static/userManual . . . 173

5 Tracciamento 174
5.1 Tracciamento Classi-Requisiti . . . 174
5.2 Tracciamento Requisiti-Classi . . . 191
5.3 Tracciamento Componenti-Requisiti . . . 219
5.4 Tracciamento Requisiti-Componenti . . . 235

Appendice A Interfaccia REST 252
A.1 Errori . . . 252
A.1.1 Errori generici . . . 252
A.1.1.1 Errori lato server . . . 252
A.1.1.2 Errori nelle richieste da parte dei client . . . 252
A.1.2 Errori specifici di Premi . . . 253
A.2 Risorse REST . . . 254

Appendice B Design patterns 263
B.1 Chain of Responsibility . . . 263
B.2 Dependecy Injection . . . 264
B.3 Facade . . . 266
B.4 Observer . . . 267

Appendice C Cytoscape 269
C.1 Organizzazione dei dati . . . 269
C.2 Inizializzazione della mappa . . . 269
C.3 Eventi . . . 269

Elenco delle figure

1 Premi::Front-End . . . 16
2 Premi::Front-End::AppConfig . . . 17
3 Premi::Front-End::AppRouter . . . 17
4 Premi::Front-End::AppRun . . . 18
5 Premi::Front-End::Controllers . . . 20
6 Premi::Front-End::Controllers::AddToPathController . . . 21
7 Premi::Front-End::Controllers::AssociationAdderController . . . 22
8 Premi::Front-End::Controllers::ContextMenuController . . . 23
9 Premi::Front-End::Controllers::DashboardController . . . 24
10 Premi::Front-End::Controllers::EditableNodeContentController . . . 26
11 Premi::Front-End::Controllers::ErrorMessageController . . . 27
12 Premi::Front-End::Controllers::HeaderController . . . 28
13 Premi::Front-End::Controllers::HierarchicalMenuController . . . 31
14 Premi::Front-End::Controllers::LoginController . . . 32
15 Premi::Front-End::Controllers::MindmapEditorController . . . 33
16 Premi::Front-End::Controllers::NodeContentsEditorController . . . 37
17 Premi::Front-End::Controllers::NodeController . . . 39
18 Premi::Front-End::Controllers::PathController . . . 40
19 Premi::Front-End::Controllers::PathsEditorController . . . 41
20 Premi::Front-End::Controllers::PathsListController . . . 44
21 Premi::Front-End::Controllers::PresentationController . . . 46
22 Premi::Front-End::Controllers::PresentationViewerController . . . 47
23 Premi::Front-End::Controllers::ProjectSettingsEditorController . . . 51
24 Premi::Front-End::Controllers::ProjectsListController . . . 52
25 Premi::Front-End::Controllers::RegistrationController . . . 53
26 Premi::Front-End::Controllers::SmartMenuController . . . 55
27 Premi::Front-End::Directives . . . 56
28 Premi::Front-End::Directives::premiAddToPath . . . 56
29 Premi::Front-End::Directives::premiAssociationAdder . . . 58
30 Premi::Front-End::Directives::premiContextMenu . . . 59
31 Premi::Front-End::Directives::premiEditableNodeContent . . . 60
32 Premi::Front-End::Directives::premiErrorMessage . . . 61
33 Premi::Front-End::Directives::premiHeader . . . 62
34 Premi::Front-End::Directives::premiHierarchicalMenu . . . 63
35 Premi::Front-End::Directives::premiMindmap . . . 64
36 Premi::Front-End::Directives::premiNode . . . 65
37 Premi::Front-End::Directives::premiNodeContent . . . 66
38 Premi::Front-End::Directives::premiNodeContentsEditor . . . 68
39 Premi::Front-End::Directives::premiPath . . . 69
40 Premi::Front-End::Directives::premiPathsList . . . 70
41 Premi::Front-End::Directives::premiPresentation . . . 71
42 Premi::Front-End::Directives::premiProjectSettingsEditor . . . 73
43 Premi::Front-End::Directives::premiProjectsList . . . 74
44 Premi::Front-End::Directives::premiSmartMenu . . . 75
45 Premi::Front-End::Model . . . 76
46 Premi::Front-End::Model::ErrorInfo . . . 77
47 Premi::Front-End::Model::Node . . . 78
48 Premi::Front-End::Model::NodeContent . . . 81
49 Premi::Front-End::Model::NodeReference . . . 83
50 Premi::Front-End::Model::Path . . . 85
51 Premi::Front-End::Model::PresentationNode . . . 87

52 Premi::Front-End::Model::Project . . . 89
53 Premi::Front-End::Services . . . 92
54 Premi::Front-End::Services::AuthenticationService . . . 92
55 Premi::Front-End::Services::MindmapAdapterService . . . 95
56 Premi::Front-End::Services::MindmapService . . . 98
57 Premi::Front-End::Services::PathService . . . 102
58 Premi::Front-End::Services::PresentationService . . . 106
59 Premi::Front-End::Services::ProjectService . . . 108
60 Premi::Front-End::Services::ServerURL . . . 111
61 Premi::Front-End::Views . . . 112
62 Premi::Front-End::Views::DashboardView . . . 112
63 Premi::Front-End::Views::LoginView . . . 113
64 Premi::Front-End::Views::MindmapEditorView . . . 114
65 Premi::Front-End::Views::PathsEditorView . . . 115
66 Premi::Front-End::Views::PresentationView . . . 116
67 Premi::Front-End::Views::RegistrationView . . . 118
68 Premi::Back-End . . . 119
69 Premi::Back-End::Server . . . 119
70 Premi::Back-End::App . . . 121
71 Premi::Back-End::App::Controllers . . . 122
72 Premi::Back-End::App::Controllers::ErrorHandler . . . 122
73 Premi::Back-End::App::Controllers::NotFoundHandler . . . 124
74 Premi::Back-End::App::Controllers::ProjectController . . . 124
75 Premi::Back-End::App::Controllers::StaticController . . . 125
76 Premi::Back-End::App::Controllers::UserController . . . 126
77 Premi::Back-End::App::Controllers::Errors . . . 126
78 Premi::Back-End::App::Controllers::Errors::PremiError . . . 127
79 Premi::Back-End::App::Controllers::Projects . . . 128
80 Premi::Back-End::App::Controllers::Projects::NodeController . . . 128
81 Premi::Back-End::App::Controllers::Projects::PathController . . . 131
82 Premi::Back-End::App::Controllers::Projects::ProjectManagementController . . . 134
83 Premi::Back-End::App::Controllers::Users . . . 137
84 Premi::Back-End::App::Controllers::Users::AuthenticationController . . . 137
85 Premi::Back-End::App::Controllers::Users::AuthorizationController . . . 139
86 Premi::Back-End::App::Models . . . 140
87 Premi::Back-End::App::Models::NodeContentModel . . . 140
88 Premi::Back-End::App::Models::NodeModel . . . 141
89 Premi::Back-End::App::Models::PathModel . . . 142
90 Premi::Back-End::App::Models::ProjectModel . . . 144
91 Premi::Back-End::App::Models::RelationModel . . . 150
92 Premi::Back-End::App::Models::Session . . . 152
93 Premi::Back-End::App::Models::UserModel . . . 152
94 Premi::Back-End::App::Routers . . . 153
95 Premi::Back-End::App::Routers::ProjectRouter . . . 154
96 Premi::Back-End::App::Routers::StaticRouter . . . 155
97 Premi::Back-End::App::Routers::UserRouter . . . 156
98 Premi::Back-End::App::Views . . . 157
99 Premi::Back-End::App::Views::UserManualView . . . 157
100 Premi::Back-End::Config . . . 157
101 Premi::Back-End::Config::Config . . . 158
102 Diagramma di Sequenza - Registrazione di un nuovo utente . . . 159
103 Diagramma di Sequenza - Apertura di un progetto . . . 160
104 Diagramma di Sequenza - Costruzione del controller . . . 161

105 Diagramma di Sequenza - Selezione di un nodo 162
106 Diagramma di Sequenza - Creazione di una presentazione 163
107 Gestione di una richiesta del client 165
108 Fallimento della procedura di registrazione 166
109 Fallimento del vincolo utente autenticato 166
110 Fallimento della richiesta di un progetto 167
111 DELETE /profile 168
112 DELETE /projects/:projectId 168
113 PUT /projects/:projectId 169
114 GET /projects/:projectId 169
115 PUT /projects/:projectId/nodes/:nodeId 170
116 POST /projects/:projectId/associations 171
117 DELETE /projects/:projectId/associations/:associationId 171
118 PUT /projects/:projectId/path/:pathId 172
119 POST /projects/:projectId/path/:pathId 172
120 GET /projects/:projectId/premiPresentation 173
121 GET /static/userManual 173
122 Chain Of Responsibility contestualizzato 264
123 Struttura logica del Dependency Injection 265
124 Facade contestualizzato 267
125 Observer contestualizzato 268

Elenco delle tabelle

1 Diario delle modifiche. 5
2 Tracciamento Classi-Requisiti 190
3 Tracciamento Requisiti-Classi 218
4 Tracciamento Componenti-Requisiti 234
5 Tracciamento Requisiti-Componenti 251

1 Introduzione

1.1 Scopo del documento

Il presente documento ha lo scopo di definire in dettaglio la struttura e il funzionamento delle componenti del *progetto*$_G$ Premi. Questo documento servirà come guida per i *Programmatori* del gruppo Pragma fornendo direttive e vincoli per la realizzazione del *progetto*$_G$.

1.2 Scopo del prodotto

Lo scopo del prodotto è di permettere la creazione e l'esecuzione di presentazioni a partire da *mappe mentali*$_G$. L'utente sarà guidato nella creazione di una *mappa mentale*$_G$ e di uno o più *percorsi di presentazione*$_G$, utilizzando i nodi di tale mappa. L'utente potrà eseguire una presentazione seguendo un *percorso* creato oppure visitando qualsiasi nodo della *mappa* costruita; rompendo così la sequenzialità nella presentazione. Il prodotto sarà utilizzabile attraverso un *browser*$_G$.

1.3 Glossario

Al fine di evitare ogni ambiguità di linguaggio e massimizzare la comprensione dei documenti, i termini tecnici, di dominio, gli acronimi e le parole che necessitano di essere chiarite, sono riportate nel documento *Glossario v2.0.0* . Ogni occorrenza dei vocaboli presenti nel *Glossario* è marcata da una "G" maiuscola in pedice ed è scritta in corsivo (es: *Esempio*$_G$).

1.4 Riferimenti

1.4.1 Normativi

- *Norme di Progetto v3.1.0*
- *Analisi dei Requisiti v2.0.0*
- **Verbale esterno:** *E5 v1.0.0* .

1.4.2 Informativi

- **Ingegneria del software - Ian Sommerville - 8a edizione (2007):**
  Parte terza: Progettazione, Capitolo 11: Progettazione architetturale, Capitolo 14: Progettazione orientata agli oggetti;
- **Design Patterns** - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides - 1a edizione italiana (2008);
- **Slide dell'insegnamento - Design patterns:**
  - Introduzione:
    http://www.math.unipd.it/~tullio/IS-1/2014/Dispense/E4.pdf;
  - Strutturali:
    http://www.math.unipd.it/~tullio/IS-1/2014/Dispense/E6.pdf;
  - Creazionali:
    http://www.math.unipd.it/~tullio/IS-1/2014/Dispense/E7.pdf;
  - Comportamentali:
    http://www.math.unipd.it/~tullio/IS-1/2014/Dispense/E8.pdf;

  - Architetturali:
    http://www.math.unipd.it/~tullio/IS-1/2014/Dispense/E9.pdf;
    http://www.math.unipd.it/~rcardin/pdf/Design%20Pattern%20Architetturali%20-%20Model%20View%20Controller_4x4.pdf;

- **Martin Fowler - *UML*$_G$ Distilled** - 2nd edition;
- **Slide dell'insegnamento - Diagrammi delle classi**: http://www.math.unipd.it/~tullio/IS-1/2014/Dispense/E2a.pdf
- **Slide dell'insegnamento - Diagrammi dei package**: http://www.math.unipd.it/~tullio/IS-1/2014/Dispense/E2b.pdf
- **Slide dell'insegnamento - Diagrammi di sequenza**: http://www.math.unipd.it/~tullio/IS-1/2014/Dispense/E3a.pdf
- **Documentazione del *framework*$_G$ *MEAN*$_G$.js**:
  http://meanjs.org/docs.html;
- **Documentazione della *piattaforma*$_G$ Node.js**:
  https://nodejs.org/documentation;
- **Breve guida ai middleware Express**:
  http://stephensugden.com/middleware_guide/;
- **Guida all'utilizzo dei middleware Express**:
  http://expressjs.com/guide/using-middleware.html;
- **Guida all'utilizzo dei middleware Passport**:
  http://passportjs.org/guide;
- **Manuale del database *MongoDB*$_G$**:
  https://docs.mongodb.org/manual;
- **Documentazione del *framework*$_G$ *AngularJS*$_G$**:
  - *Documentazione generica*: https://docs.angularjs.org/guide;
  - *Documentazione servizio $http*: https://docs.angularjs.org/api/ng/service/$http;
  - *Documentazione servizio $location*: https://docs.angularjs.org/api/ng/service/$location;
  - *Documentazione servizio $window*: https://docs.angularjs.org/api/ng/service/$window;
  - *Documentazione servizio $routeParams*: https://docs.angularjs.org/api/ngRoute/service/$routeParams;
  - *Documentazione servizio $q*: https://docs.angularjs.org/api/ng/service/$q.
- **Documentazione del *framework*$_G$ Material for Angular**:
  https://material.angularjs.org/latest/#/.

2 Specifica del Front-End

2.1 Premi::Front-End

2.1.1 Informazioni generali

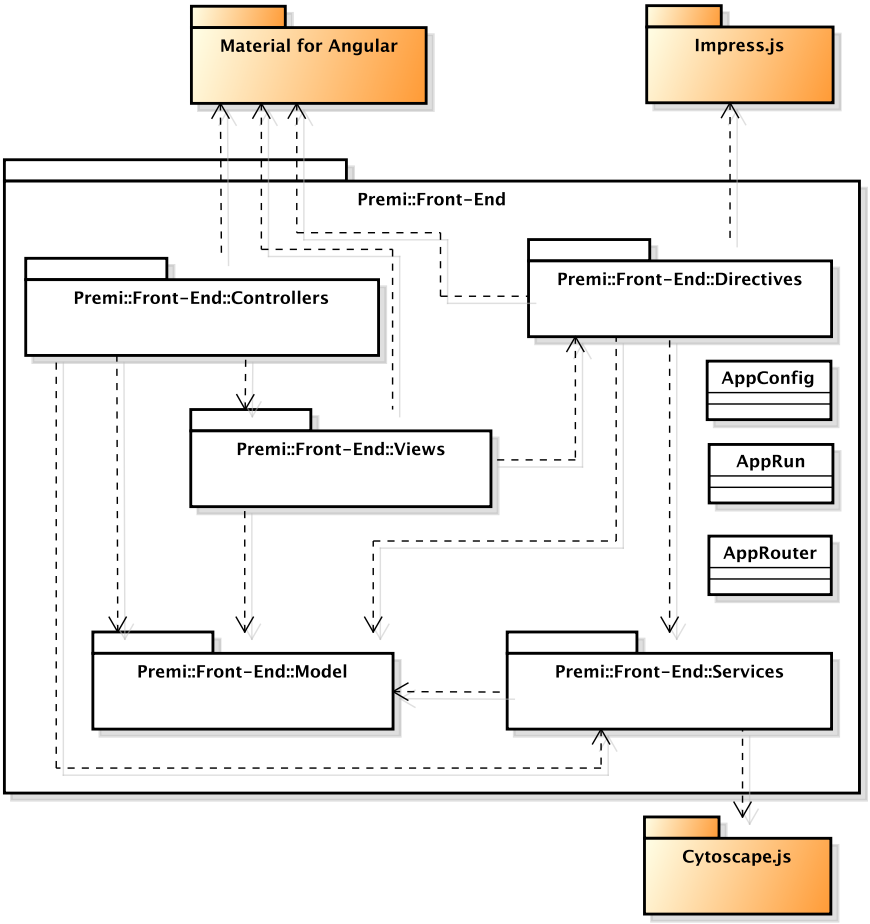


Figura 1: Premi::Front-End

- **Descrizione**
  Package contenente le varie componenti della parte front end dell'applicazione.
- **Package contenuti**:
  - `Controllers`
    Package contenente i controller della componente front-end dell'applicazione.
  - `Directives`
    Package contenente le directives che compongno le views.
  - `Model`
    Package contenente le classi che definiscono la business logic dell'applicazione.
  - `Services`
    Package contenente le classi che descrivono il meccanismo con cui il front-end può interfacciarsi con le API REST del back-end. Permette di popolare il model del client e di richiedere l'esecuzione di operazioni al server.

- `Views`
  Package contenente le views della componente front-end dell'applicazione.

2.1.2 Classi

2.1.2.1 Premi::Front-End::AppConfig

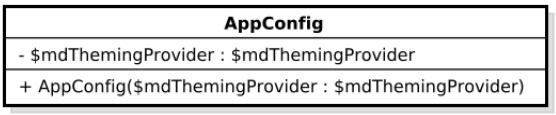


Figura 2: Premi::Front-End::AppConfig

- **Descrizione**
  Classe che si occupa di definire la configurazione dell'applicazione.
- **Utilizzo**
  Viene utilizzata per definire lo schema di colori per l'applicazione.
- **Attributi**:
  - `- $mdThemingProvider: $mdThemingProvider`
    Campo dati contenente un riferimento al servizio della libreria *Material for Angular* che permette di definire lo schema colori dell'applicazione.
- **Metodi**:
  - `+ AppConfig($mdThemingProvider: $mdThemingProvider)`
    Metodo costruttore della classe.
    Viene utilizzato per impostare i colori di default dell'applicazione.
    Parametri:
    * `$mdThemingProvider: $mdThemingProvider`
      Parametro contenente un riferimento al servizio della libreria *Material for Angular* che permette di definire lo schema colori dell'applicazione.

2.1.2.2 Premi::Front-End::AppRouter

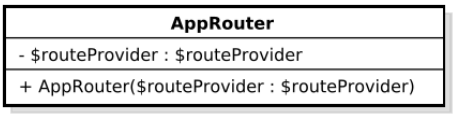


Figura 3: Premi::Front-End::AppRouter

- **Descrizione**
  Classe che gestisce i routes dell'applicazione, utilizza il servizio `$routeProvider` per associare ad ogni route un controller e una view.
- **Utilizzo**
  Viene utilizzata per associare un URL alle varie view dell'applicazione.
- **Attributi**:

- `- $routeProvider: $routeProvider`
  Campo dati contenente un riferimento al servizio di Angular che si occupa di definire le route per l'applicazione.

- **Metodi**:
  - `+ AppRouter($routeProvider: $routeProvider)`
    Metodo costruttore della classe.
    Parametri:
    * `$routeProvider: $routeProvider`
      Parametro contenente un riferimento al servizio di Angular che si occupa di definire le route per l'applicazione.

2.1.2.3 Premi::Front-End::AppRun

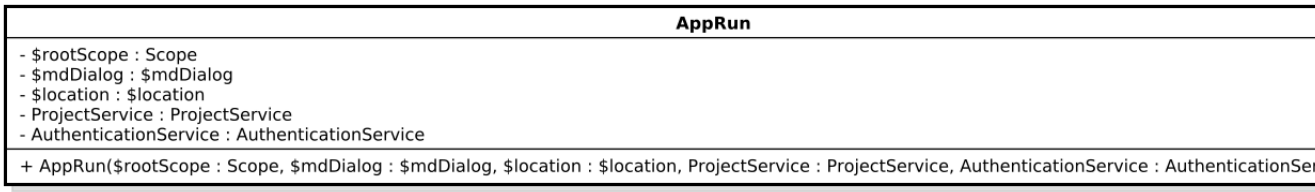


Figura 4: Premi::Front-End::AppRun

- **Descrizione**
  Classe che si occupa di gestire l'inizializzazione dell'applicazione.

- **Utilizzo**
  Viene utilizzata per registrare i gestori per gli eventi globali quali: `premi-fullscreen-on`/`premi-fullscreen-off` per la gestione della modalità a schermo intero e `premi-error` per la gestione dei messaggi d'errore. Questa classe viene anche utilizzata per verificare che l'utente sia autenticato e che ci sia un progetto aperto, prima della visualizzazione delle varie view.

- **Relazioni con altre classi**:
  - *OUT* `premiErrorMessage`
    Rappresenta il componente grafico che permette di mostrare messaggi d'errore all'utente all'interno dell'applicazione. Questo componente fornisce anche un pulsante che permette di nascondere il messaggio.
  - *OUT* `ErrorInfo`
    Rappresenta le informazioni di un errore che si è verificato eseguendo una determinata operazione.
  - *OUT* `AuthenticationService`
    Questa classe si occupa di gestire il processo di autenticazione e di registrazione di un utente.
  - *OUT* `ProjectService`
    Questa classe si occupa del recupero e della modifica delle informazioni riguardanti i progetti.

- **Attributi**:

- `- AuthenticationService: AuthenticationService`
  Campo dati contenente un riferimento al servizio che si occupa della gestione delle informazioni legate all'autenticazione.
- `- ProjectService: ProjectService`
  Campo dati contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
- `- $location: $location`
  Campo dati contenente un riferimento al servizio creato da *Angular*$_G$ che permette di accedere alla barra degli indirizzi del *browser*$_G$, i cambiamenti all'URL nella barra degli indirizzi si riflettono in questo oggetto e viceversa.
- `- $mdDialog: $mdDialog`
  Campo dati contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
- `- $rootScope: Scope`
  Campo dati contenente un riferimento all'oggetto `$rootScope` creato da Angular è lo `$scope` principale dell'applicazione, tutti gli altri `$scope` discendono da questo oggetto.

- **Metodi**:
  - `+ AppRun($rootScope: Scope, $location: $location, $mdDialog: $mdDialog, ProjectService: ProjectService, AuthenticationService: AuthenticationService)`
    Metodo costruttore della classe.
    Parametri:
    * `$rootScope: Scope`
      Parametro contenente un riferimento all'oggetto `$rootScope` creato da *Angular*$_G$, è lo `$scope` principale dell'applicazione, tutti gli altri `$scope` discendono da questo oggetto.
    * `$location: $location`
      Parametro contenente un riferimento al servizio creato da *Angular*$_G$ che permette di accedere alla barra degli indirizzi del *browser*$_G$, i cambiamenti all'URL nella barra degli indirizzi si riflettono in questo oggetto e viceversa.
    * `$mdDialog: $mdDialog`
      Parametro contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
    * `ProjectService: ProjectService`
      Parametro contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
    * `AuthenticationService: AuthenticationService`
      Parametro contenente un riferimento al servizio che si occupa della gestione delle informazioni legate all'autenticazione.

2.2 Premi::Front-End::Controllers

2.2.1 Informazioni generali

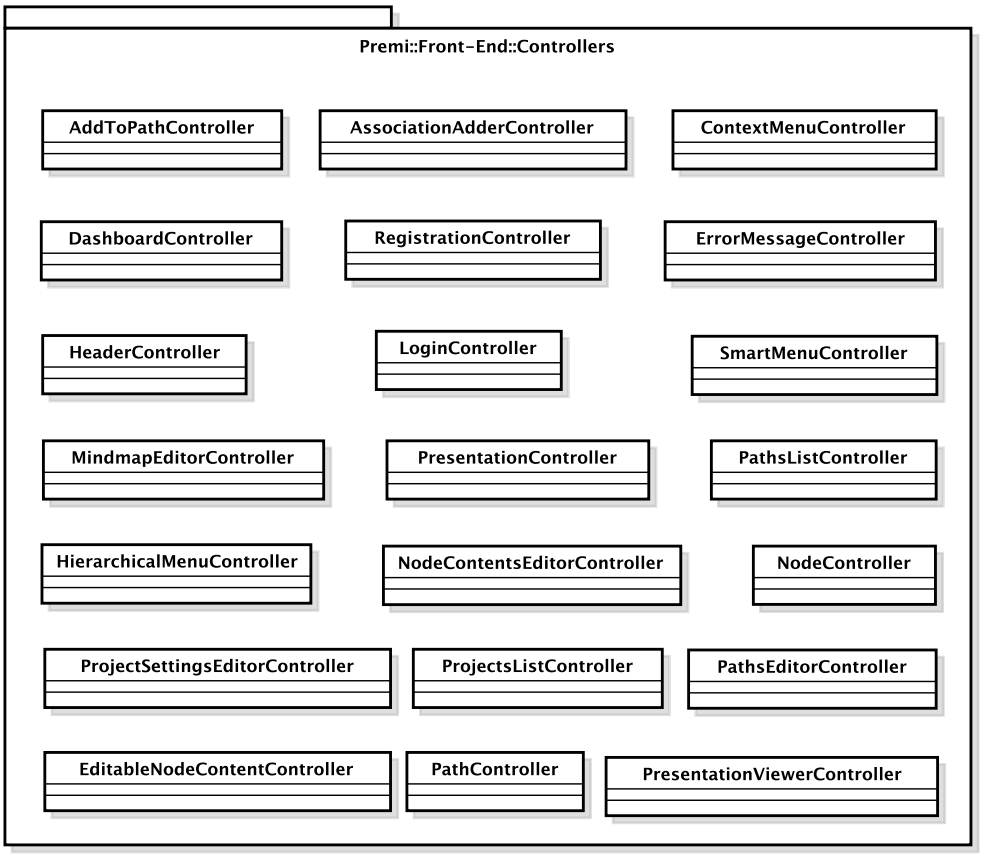


Figura 5: Premi::Front-End::Controllers

- **Descrizione**
  Package contenente i controller della componente front-end dell'applicazione.
- **Padre**: `Front-End`
- **Interazioni con altri componenti**:
  - `Directives`
    Package contenente le directives che compongno le views.
  - `Model`
    Package contenente le classi che definiscono la business logic dell'applicazione.
  - `Services`
    Package contenente le classi che descrivono il meccanismo con cui il front-end può interfacciarsi con le API REST del back-end. Permette di popolare il model del client e di richiedere l'esecuzione di operazioni al server.
  - `Views`
    Package contenente le views della componente front-end dell'applicazione.

2.2.2 Classi

2.2.2.1 Premi::Front-End::Controllers::AddToPathController

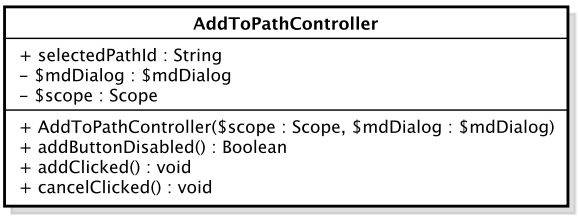


Figura 6: Premi::Front-End::Controllers::AddToPathController

- **Descrizione**
  Classe che si occupa di gestire la logica di funzionamento della directive `premiAddToPath`.
- **Utilizzo**
  Viene utilizzata per permettere all'utente di aggiungere nodi ai percorsi di presentazione.
- **Relazioni con altre classi**:
  - *OUT* `premiAddToPath`
    Rappresenta il componente grafico che permette all'utente di visualizzare il contenuto di un nodo e di aggiungerlo ad uno dei percorsi di presentazione esistenti. Questo componente visualizza una lista contenente tutti i percorsi disponibili al quale, selezionando quello desiderato, è possibile aggiungere il nodo slezionato.
- **Attributi**:
  - `+ selectedPathId: String`
    Campo dati contenente l'`id` del percorso di presentazione selezionato dall'utente.
  - `- $mdDialog: $mdDialog`
    Campo dati contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
  - `- $scope: Scope`
    Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.
- **Metodi**:
  - `+ addButtonDisabled(): Boolean`
    Metodo che ritorna un valore booleano che specifica se il pulsante di conferma è abilitato o meno.
  - `+ addClicked(): void`
    Metodo che gestisce l'evento `click` sul pulsante per confermare l'aggiunta del nodo al percorso di presentazione.
  - `+ AddToPathController($scope: Scope, $mdDialog: $mdDialog)`
    Metodo costruttore della classe.
    Parametri:
    * `$scope: Scope`
      Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
    * `$mdDialog: $mdDialog`
      Parametro contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.

- `+ cancelClicked(): void`
  Metodo che gestisce l'evento `click` sul pulsante che annulla l'aggiunta del nodo al percorso di presentazione.

2.2.2.2 Premi::Front-End::Controllers::AssociationAdderController

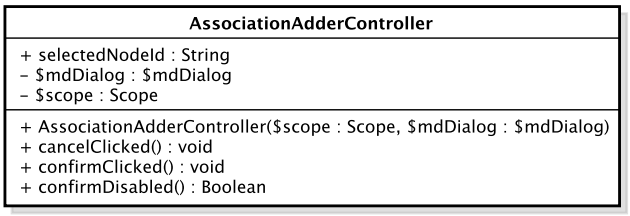


Figura 7: Premi::Front-End::Controllers::AssociationAdderController

- **Descrizione**
  Classe che gestisce il funzionamento della directive `premiAssociationAdder`.
- **Utilizzo**
  Viene utilizzata per consentire all'utente di aggiungere associazioni tra i nodi della mappa mentale.
- **Relazioni con altre classi**:
  - *OUT* `premiAssociationAdder`
    Rappresenta il componente grafico che permette all'utente di creare un'associazione tra il nodo selezionato e un altro nodo della mappa.
- **Attributi**:
  - `+ selectedNodeId: String`
    Campo dati contenente l'`id` dell'oggetto `NodeReference` selezionato dall'utente.
  - `- $mdDialog: $mdDialog`
    Campo dati contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
  - `- $scope: Scope`
    Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.
- **Metodi**:
  - `+ AssociationAdderController($scope: Scope, $mdDialog: $mdDialog)`
    Metodo costruttore della classe.
    Parametri:
    * `$scope: Scope`
      Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
    * `$mdDialog: $mdDialog`
      Campo dati contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.

- `+ cancelClicked(): void`
  Metodo che gestisce l'evento click sul pulsante per annullare l'aggiunta dell'associazione. Si occupa di nascondere la directive.
- `+ confirmClicked(): void`
  Metodo che gestisce l'evento di click sul pulsante di conferma di aggiunta dell'associazione. Questo metodo invoca la funzione `$scope.onNodeSelected` e nasconde la directive.
- `+ confirmDisabled(): Boolean`
  Metodo che ritorna un booleano che specifica se il pulsante di conferma è abilitato o meno.

2.2.2.3 Premi::Front-End::Controllers::ContextMenuController

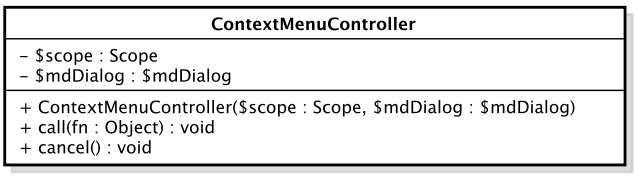


Figura 8: Premi::Front-End::Controllers::ContextMenuController

- **Descrizione**
  Classe che gestisce il comportamento della directive `premiContextMenu`.
- **Utilizzo**
  Viene utilizzata per gestire il menù che compare quando l'utente seleziona un nodo o un'associazione della mappa mentale.
- **Relazioni con altre classi**:
  - *OUT* `premiContextMenu`
    Rappresenta un menù contestuale generato in base agli oggetti passati nello scope isolato. Fornisce un pulsante per ogni oggetto ricevuto come parametro, ogni pulsante viene rappresentato con un'icona e con del testo. Al click di un pulsante viene invocata la funzione ad esso associata.
  - *OUT* `Node`
    Rappresenta un nodo della mappa mentale. Contiene tutte le informazioni necessarie alla presentazione del contenuto del nodo.
- **Attributi**:
  - `- $mdDialog: $mdDialog`
    Campo dati contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
  - `- $scope: Scope`
    Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.
- **Metodi**:
  - `+ call(fn: Object): void`
    Metodo che invoca la funzione la funzione `callback` presente all'interno dell'oggetto `fn` ricevuto come parametro.
    Parametri:

    * `fn: Object`
      Parametro che rappresenta una voce del menù, ha un campo dati `callback` contenente una funzione da invoca quando l'utente seleziona la voce del menù.

  - `+ cancel(): void`
    Metodo che nasconde il menù utilizzando il metodo `$mdDialog.cancel`.
  - `+ ContextMenuController($scope: Scope, $mdDialog: $mdDialog)`
    Metodo costruttore della classe.
    Parametri:
    * `$scope: Scope`
      Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
    * `$mdDialog: $mdDialog`
      Parametro contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.

2.2.2.4 Premi::Front-End::Controllers::DashboardController

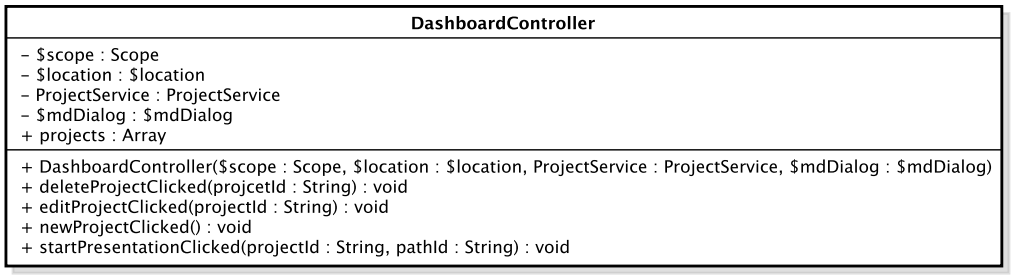


Figura 9: Premi::Front-End::Controllers::DashboardController

- **Descrizione**
  Classe che gestisce le operazioni e la logica applicativa riguardante la visualizzazione dei progetti di un utente.
- **Utilizzo**
  Viene utilizzata per consentire all'utente di visualizzare tutti i propri progetti creati e crearne di nuovi. Dalla lista dei progetti presenti è possibile selezionare un progetto per rinominarlo oppure aprirlo, in modo da poterlo modificare e/o presentare.
- **Relazioni con altre classi**:
  - *OUT* `ProjectService`
    Questa classe si occupa del recupero e della modifica delle informazioni riguardanti i progetti.
  - *OUT* `DashboardView`
    View dell'applicazione contenente la lista dei progetti dell'utente correntemente autenticato. Da questa view è possibile:
    * Creare un nuovo progetto;
    * Aprire un progetto esistente per modificarlo oppure presentarlo;
    * Cancellare un progetto esistente.

- **Attributi**:
  - `+ projects: Array`
    Campo dati contenente i progetti creati dall'utente.
  - `- ProjectService: ProjectService`
    Campo dati contenente un riferimento al servizio per la gestione dei progetti.
  - `- $location: $location`
    Campo dati contenente un riferimento al servizio creato da *Angular*$_G$ che permette di accedere alla barra degli indirizzi del *browser*$_G$, i cambiamenti all'URL nella barra degli indirizzi si riflettono in questo oggetto e viceversa.
  - `- $mdDialog: $mdDialog`
    Campo dati contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
  - `- $scope: Scope`
    Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.
- **Metodi**:
  - `+ DashboardController($scope: Scope, ProjectService: ProjectService, $location: $location, $mdDialog: $mdDialog)`
    Metodo costruttore della classe.
    Parametri:
    * `$scope: Scope`
      Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
    * `ProjectService: ProjectService`
      Parametro che rappresenta il riferimento al servizio di gestione dei progetti.
    * `$location: $location`
      Parametro contenente un riferimento al servizio creato da *Angular*$_G$ che permette di accedere alla barra degli indirizzi del *browser*$_G$, i cambiamenti all'URL nella barra degli indirizzi si riflettono in questo oggetto e viceversa.
    * `$mdDialog: $mdDialog`
      Parametro contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
  - `+ deleteProjectClicked(projectId: String): void`
    Metodo che gestisce l'evento di cancellazione di un progetto esistente. Interagisce con `ProjectService` per cancellare il progetto.
    Parametri:
    * `projectId: String`
      Parametro contenente l'id del progetto da cancellare.
  - `+ editProjectClicked(projectId: String): void`
    Metodo che si occupa di reindirizzare l'utente alla vista per la modifica di un progetto.
    Parametri:
    * `projectId: String`
      Parametro contenente l'id del progetto da aprire.
  - `+ newProjectClicked(): void`
    Metodo che gestisce l'evento di creazione di un nuovo progetto. Interagisce con `ProjectService` per creare un nuovo progetto.

- `+ startPresentationClicked(pathId: String, projectId: String): void`
  Metodo che gestisce l'avvio di una presentazione direttamente dalla dashboard. Instanzia il progetto e reindirizza alla vista per la presentazione.
  Parametri:
  * `pathId: String`
    Parametro contenente l'id del percorso di presentazione che l'utente vuole presentare.
  * `projectId: String`
    Parametro contenente l'id del progetto contenente il percorso che l'utente vuole presentare.

2.2.2.5 Premi::Front-End::Controllers::EditableNodeContentController

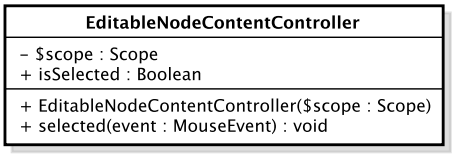


Figura 10: Premi::Front-End::Controllers::EditableNodeContentController

- **Descrizione**
  Classe che gestisce la logica della directive `premiEditableNodeContent`. Questa classe registra un gestore per l'evento `nodecontent-deselect` che imposta come de-selezionata la directive.
- **Utilizzo**
  Viene utilizzata per gestire la selezione e de-selezione del contenuto di un nodo da parte dell'utente.
- **Relazioni con altre classi**:
  - *OUT* `premiEditableNodeContent`
    Rappresenta un elemento contenuto nel frame di un nodo. Questo elemento è spostabile e ridimensionabile. Le modifiche effettuate su questo elemento vengono riportate direttamente sull'oggetto `nodeContent`. Questo componente deve essere in grado di rappresentare le varie tipologie di contenuto che possono essere presenti all'interno di un nodo. La discriminazione del tipo del contenuto deve essere fatta in base ai campi dati dell'oggetto `nodeContent`.
- **Attributi**:
  - `+ isSelected: Boolean`
    Campo dati che specifica se l'elemento è stato selezionato dall'utente o meno.
  - `- $scope: Scope`
    Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.
- **Metodi**:
  - `+ EditableNodeContentController($scope: Scope)`
    Metodo costruttore della classe.
    Parametri:

    * `$scope: Scope`
      Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.

  - `+ selected(event: MouseEvent): void`
    Metodo che gestisce la selezione dell'elemento da parte dell'utente. Questo metodo solleva l'evento `nodecontent-selected` fornendo l'`id` dell'oggetto `NodeContent` rappresentato dalla directive.
    Parametri:
    * `event: MouseEvent`
      Parametro contenente le informazioni associate all'evento del browser che ha portato all'invocazione del metodo.

2.2.2.6 Premi::Front-End::Controllers::ErrorMessageController

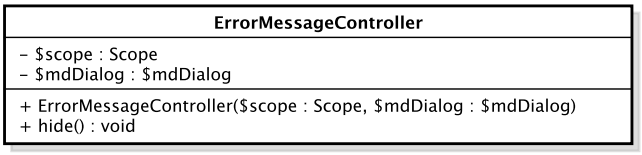


Figura 11: Premi::Front-End::Controllers::ErrorMessageController

- **Descrizione**
  Classe che gestisce la logica di visualizzazione della directive `premiErrorMessage`.
- **Utilizzo**
  Viene utilizzata per nascondere la directive quando l'utente prende l'apposito pulsante.
- **Relazioni con altre classi**:
  - *OUT* `premiErrorMessage`
    Rappresenta il componente grafico che permette di mostrare messaggi d'errore all'utente all'interno dell'applicazione. Questo componente fornisce anche un pulsante che permette di nascondere il messaggio.
- **Attributi**:
  - `- $mdDialog: $mdDialog`
    Campo dati contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
  - `- $scope: Scope`
    Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.
- **Metodi**:
  - `+ ErrorMessageController($scope: Scope, $mdDialog: $mdDialog)`
    Metodo costruttore della classe.
    Parametri:

  * `$scope: Scope`
    Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
  * `$mdDialog: $mdDialog`
    Parametro contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
- `+ hide(): void`
  Metodo che nasconde la directive.

2.2.2.7 Premi::Front-End::Controllers::HeaderController

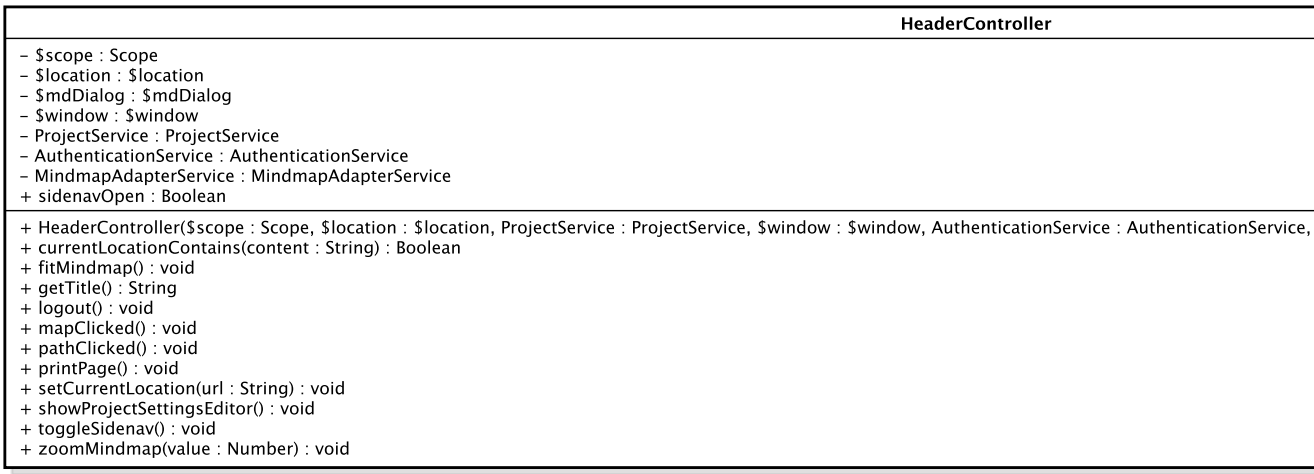


Figura 12: Premi::Front-End::Controllers::HeaderController

- **Descrizione**
  Classe che gestisce le operazioni e la logica applicativa della directive `premiHeader`,
- **Utilizzo**
  Viene utilizzata per consentire all'utente di spostarsi tra le varie views dell'applicazione.
- **Relazioni con altre classi**:
  - *OUT* `premiHeader`
    Rappresenta la barra di navigazione che permette all'utente di spostarsi tra le varie views dell'applicazione. Fornisce inoltre i pulsanti per visualizzare il manuale utente, effettuare il logout, chiudere al presentazione e il progetto. Inoltre, quando l'utente si trova nelle views in cui è possibile modificare il progetto, questa directive permette di mostrare la finestra di modifica delle impostazioni del progetto, sia di interagire con la visualizzazione della mappa mentale, modificando lo zoom e permettendo anche di stamparla.
- **Attributi**:
  - `- AuthenticationService: AuthenticationService`
    Campo dati contenente un riferimento al servizio che si occupa della gestione delle informazioni legate all'autenticazione.
  - `- MindmapAdapterService: MindmapAdapterService`
    Campo dati contenente un riferimento al servizio `MindmapAdapterService`

- `- ProjectService: ProjectService`
  Campo dati contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
- `+ sidenavOpen: Boolean`
  Campo dati che specifica se il menù laterale è aperto o meno.
- `- $location: $location`
  Campo dati contenente un riferimento al servizio creato da *Angular*$_G$ che permette di accedere alla barra degli indirizzi del *browser*$_G$, i cambiamenti all'URL nella barra degli indirizzi si riflettono in questo oggetto e viceversa.
- `- $mdDialog: $mdDialog`
  Campo dati contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
- `- $scope: Scope`
  Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.
- `- $window: $window`
  Campo dati contenente un riferimento al servizio creato da *Angular*$_G$ che permette di accedere alle proprietà della finestra del browser.

- **Metodi**:
  - `+ currentLocationContains(content: String): Boolean`
    Metodo che controlla se l'URL corrente contiene la stringa ricevuta come parametro.
    Parametri:
    * `content: String`
      Parametro che rappresenta il contenuto da cercare.
  - `+ fitMindmap(): void`
    Metodo che tramite `MindmapAdapterService` adatta le dimensioni della mappa mentale a quelle dello schermo.
  - `+ getTitle(): String`
    Metodo che, in base alla view corrente, ritorna il titolo da visualizzare.
  - `+ HeaderController($scope: Scope, $location: $location, ProjectService: ProjectService, $window: $window, AuthenticationService: AuthenticationService, MindmapAdapterService: MindmapAdapterService, $mdDialog: $mdDialog)`
    Metodo costruttore della classe.
    Parametri:
    * `$scope: Scope`
      Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
    * `$location: $location`
      Parametro contenente un riferimento al servizio creato da *Angular*$_G$ che permette di accedere alla barra degli indirizzi del *browser*$_G$, i cambiamenti all'URL nella barra degli indirizzi si riflettono in questo oggetto e viceversa.
    * `ProjectService: ProjectService`
      Parametro contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
    * `$window: $window`
      Parametro contenente un riferimento al servizio creato da *Angular*$_G$ che permette di accedere alle proprietà della finestra del browser.

    * `AuthenticationService: AuthenticationService`
      Parametro contenente un riferimento al servizio che si occupa della gestione delle informazioni legate all'autenticazione.
    * `MindmapAdapterService: MindmapAdapterService`
      Paramentro contenente un riferimento al servizio `MindmapAdapterService`
    * `$mdDialog: $mdDialog`
      Parametro contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.

- `+ logout(): void`
  Metodo che gestisce l'evento `click` sul pulsante per il logout.
- `+ mapClicked(): void`
  Metodo che gestisce l'evento di click sul pulsante per passare alla view che permette di modificare la mappa mentale, reindirizzando l'utente all'opportuna view.
- `+ pathClicked(): void`
  Metodo che gestisce l'evento di click sul pulsante per passare alla view che permette di modificare i percorsi di presentazione, reindirizzando l'utente all'opportuna view.
- `+ printPage(): void`
  Metodo che, utilizzando il servizio `$window`, invoca la funzionalità di stampa del browser.
- `+ setCurrentLocation(url: String): void`
  Metodo che reindirizza l'utente alla view identificata dall'URL ricevuto come parametro.
  Parametri:
    * `url: String`
      Parametro che rappresenta l'URL da utilizzare per il reindirizzamento.
- `+ showManual(ev: MouseEvent): void`
  Metodo che visualizza il pop-up contenente il manuale utente.
  Parametri:
    * `ev: MouseEvent`
      Parametro contenente le informazioni relative all'evento del browser che ha portato all'invocazione del metodo.
- `+ showProjectSettingsEditor(): void`
  Metodo che utilizzando il servizio `$mdDialog` visualizza la directive per la modifica delle impostazioni del progetto sotto forma di finestra a pop-up.
- `+ toggleSidenav(): void`
  Metodo che inverte il valore del campo dati `sidenavOpen`.
- `+ zoomMindmap(value: Number): void`
  Metodo che regola lo zoom della mappa mentale, usa `MindmapAdapterService` per effettuare il ridimensionamento della mappa.
  Parametri:
    * `value: Number`
      Parametro che rappresenta la quantità di incremento o di decremento dello zoom.

2.2.2.8 Premi::Front-End::Controllers::HierarchicalMenuController

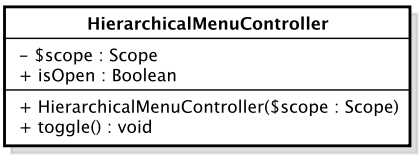


Figura 13: Premi::Front-End::Controllers::HierarchicalMenuController

- **Descrizione**
  Classe che gestisce la visibilità della directive `premiHierarchicalMenu`.
- **Utilizzo**
  Viene utilizzata per permettere all'utente di nascondere il menù con i nodi in relazione al nodo visualizzato in presentazione.
- **Relazioni con altre classi**:
  - *OUT* `premiHierarchicalMenu`
    Rappresenta il menù che durante la presentazione permette di navigare la mappa mentale seguendo le relazioni gerarchiche presenti tra i vari nodi della mappa mentale. Questo componente consiste in una lista di elementi selezionabili dall'utente, ogni elemento corrisponde ad una relazione presente tra il nodo corrente ed un altro nodo della mappa.
- **Attributi**:
  - `+ isOpen: Boolean`
    Campo dati che specifica se il menù è aperto o chiuso.
  - `- $scope: Scope`
    Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.
- **Metodi**:
  - `+ HierarchicalMenuController($scope: Scope)`
    Metodo costruttore della classe.
    Parametri:
    * `$scope: Scope`
      Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
  - `+ toggle(): void`
    Metodo che inverte il valore del campo dati `isOpen`

2.2.2.9 Premi::Front-End::Controllers::LoginController

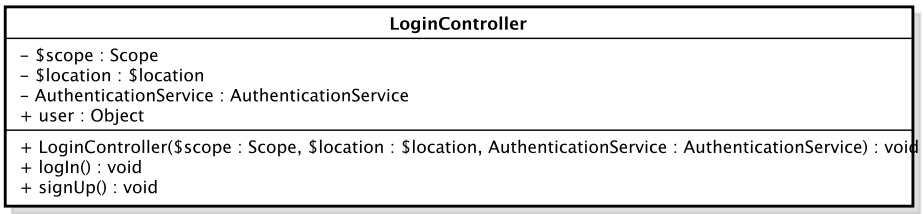


Figura 14: Premi::Front-End::Controllers::LoginController

- **Descrizione**
  Classe che gestisce le operazioni e la logica applicativa riguardante la pagina di login.
- **Utilizzo**
  Viene utilizzata per permettere all'utente di effettuare l'accesso all'applicazione. Prima della creazione della view viene verificato che non sia già presente una sessione, in caso affermativo l'utente viene reindirizzato alla lista dei progetti disponibili. Se non c'è nessuna sessione attiva, viene creata la view che fornisce all'utente il form necessario per effettuare il login. Il controllo della sessione e l'autenticazione verranno eseguiti sfruttando `AuthenticationService`.
- **Relazioni con altre classi**:
  - *OUT* `AuthenticationService`
    Questa classe si occupa di gestire il processo di autenticazione e di registrazione di un utente.
  - *OUT* `LoginView`
    View dell'applicazione contenente il form necessario per effettuare il login. Fornisce inoltre un link alla pagina di registrazione.
- **Attributi**:
  - `- AuthenticationService: AuthenticationService`
    Campo dati contenente un riferimento al servizio che si occupa della gestione delle informazioni legate all'autenticazione.
  - `+ user: Object`
    Campo dati contenente un oggetto con i seguenti attributi: `email` e `password`.
  - `- $location: $location`
    Campo dati contenente un riferimento al servizio creato da *Angular*$_G$ che permette di accedere alla barra degli indirizzi del *browser*$_G$, i cambiamenti all'URL nella barra degli indirizzi si riflettono in questo oggetto e viceversa.
  - `- $scope: Scope`
    Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.
- **Metodi**:
  - `+ logIn(): void`
    Metodo che gestisce l'evento click sul pulsante di login. Recupera i dati dallo $scope e, mediante `AuthenticationService`, prova ad effettuare l'autenticazione. Se l'autenticazione ha esito positivo, reindirizza l'utente alla dashboard, altrimenti visualizza un messaggio d'errore.
  - `+ LoginController($scope: Scope, $location: $location, AuthenticationService: AuthenticationService)`

Metodo costruttore della classe.
Parametri:

* `$scope: Scope`
  Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
* `$location: $location`
  Parametro contenente un riferimento al servizio creato da *Angular*$_G$ che permette di accedere alla barra degli indirizzi del *browser*$_G$, i cambiamenti all'URL nella barra degli indirizzi si riflettono in questo oggetto e viceversa.
* `AuthenticationService: AuthenticationService`
  Parametro contenente un riferimento al servizio che si occupa della gestione delle informazioni legate all'autenticazione.

– `+ signUp(): void`
Metodo che gestisce l'evento click sul pulsante di registrazione. Effettua il redirect alla pagina di registrazione.

2.2.2.10 Premi::Front-End::Controllers::MindmapEditorController

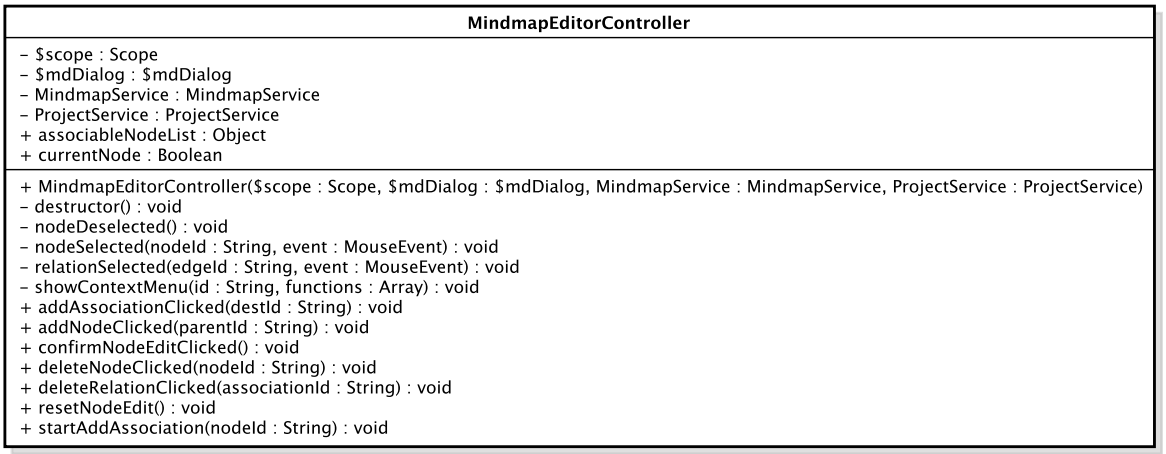


Figura 15: Premi::Front-End::Controllers::MindmapEditorController

- **Descrizione**
  Classe che gestisce le operazioni e la logica applicativa riguardante la modifica di una mappa.
- **Utilizzo**
  Questa classe viene utilizzata per consentire all'utente di modificare la mappa. In particolare, tramite `EditMapView` e le directives che la compongono, permette all'utente di:
  – Aggiungere/modificare/eliminare nodi;
  – Aggiungere/eliminare associazioni tra i nodi;
  – Modificare le impostazioni del progetto.

  Tutte queste modifiche vengono effettuate utilizzando `MindmapService`.
- **Relazioni con altre classi**:

- *OUT* `Node`
  Rappresenta un nodo della mappa mentale. Contiene tutte le informazioni necessarie alla presentazione del contenuto del nodo.
- *OUT* `Project`
  Rappresenta un progetto creato dall'utente. Contiene le informazioni riguardanti i parametri globali del progetto: nome, formato del testo, colore di sfondo.
- *OUT* `MindmapService`
  Questa classe si occupa del recupero e della modifica delle informazioni relative ai nodi e alle associazione presenti nella mappa mentale del progetto corrente. Utilizza `$http` per comunicare con il back-end e `MindmapAdapterService` per memorizzare i dati in locale.
- *OUT* `ProjectService`
  Questa classe si occupa del recupero e della modifica delle informazioni riguardanti i progetti.
- *OUT* `MindmapEditorView`
  View dell'applicazione contenente la mappa mentale. Offre funzionalità di modifica della mappa. Da questa view è possibile:
  * Aggiungere/togliere nodi;
  * Modificare il contenuto di un nodo;
  * Aggiungere/togliere associazioni tra nodi;
  * Modificare i parametri del progetto.

- **Attributi**:
  - `+ associableNodesList: Object`
    Campo dati contenente le informazioni riguardanti i nodi della mappa mentale che possono essere associati al `currentNode`. Le informazioni all'interno dell'oggetto sono memorizzate come coppie chiave/valore, usando come chiave l'`id` del nodo e come valore il titolo.
  - `+ currentNode: Node`
    Campo dati sul quale effettuare le modifiche temporanee prima che l'utente le confermi. Viene usato anche per contenere i dati del nodo da visualizzare in anteprima.
  - `- MindmapService: MindmapService`
    Campo dati contenente un riferimento al servizio che si occupa della gestione della mappa mentale.
  - `- ProjectService: ProjectService`
    Campo dati contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
  - `- $mdDialog: $mdDialog`
    Campo dati contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
  - `- $scope: Scope`
    Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.

- **Metodi**:
  - `+ addAssociationClicked(destId: String): void`
    Metodo che gestisce l'evento di creazione di un'associazione tra due nodi. Per creare l'associazione è necessario utilizzare `AssociationService`, l'informazione riguardante l'origine è disponibile nell'oggetto `$scope.currentNode`.
    Parametri:

    * `destId: String`
      Parametro contenente l'id del nodo destinazione dell'associazione.

- `+ addNodeClicked(parentId: String): void`
  Metodo che gestisce l'evento di aggiunta di un nuovo nodo alla mappa mentale. Utilizza `MindmapService` per aggiungere un nodo vuoto alla mappa mentale, come figlio del nodo passato per parametro.
  Parametri:
    * `parentId: String`
      Parametro contenente l'id del nodo al quale bisogna aggiungere un nuovo figlio.
- `+ confirmNodeEditClicked(): void`
  Metodo che gestisce l'evento di conferma della modifica di un nodo. Rende permanenti le modifiche subite dall'oggetto nodo presente nello `$scope`. Utilizza `MindmpaService`.
- `+ deleteNodeClicked(nodeId: String): void`
  Metodo che gestisce l'evento di cancellazione di un nodo. Utilizza `MindmapService` per cancellare il nodo dalla mappa mentale.
  Parametri:
    * `nodeId: String`
      Parametro contenente l'id del nodo da cancellare.
- `+ deleteRelationClicked(associationId: String): void`
  Metodo che gestisce l'evento di cancellazione di un'associazione. Utilizza `MindmapService` per cancellare l'associazione dato l'id.
  Parametri:
    * `associationId: String`
      Parametro contenente l'id dell'associazione da cancellare.
- `- destructor(): void`
  Metodo che gestisce l'evento `destroy` del controller, viene utilizzato per interrompere l'ascolto degli eventi di `MindmapService`.
- `+ editNodeClicked(): void`
  Metodo che gestisce l'evento `click` sul pulsante che permette di modificare il contenuto del nodo presente nel campo dati `currentNode`.
- `+ MindmapEditorController($scope: Scope, $mdDialog: $mdDialog, MindmapService: MindmapService, ProjectService: ProjectService)`
  Metodo costruttore della classe.
  Parametri:
    * `$scope: Scope`
      Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
    * `$mdDialog: $mdDialog`
      Parametro contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
    * `MindmapService: MindmapService`
      Parametro contenente un riferimento al servizio che si occupa della gestione della mappa mentale.
    * `ProjectService: ProjectService`
      Parametro contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
- `- nodeDeselected(): void`
  Metodo che deseleziona il nodo corrente.

- `- nodeSelected(e: MouseEvent, nodeId: String): void`
  Metodo che gestisce l'evento di selezione di un nodo, viene invocato come gestore dell'evento sollevato da `MindmapService`. Si occupa di aggiornare lo `$scope` con i dati del nodo selezionato dall'utente e di visualizzare il menù contestuale in modo da permettere all'utente di modificare il nodo.
  Parametri:
  * `e: MouseEvent`
    Parametro contenente i dati dell'evento click sollevato dal browser.
  * `nodeId: String`
    Parametro contenente l'id del nodo selezionato dall'utente.
- `- relationSelected(e: MouseEvent, edgeId: String): void`
  Metodo che gestisce l'evento di selezione di una relazione, viene invocato come gestore dell'evento sollevato da `MindmapService`. Si occupa di aggiornare i dati del menù di modifica di una relazione e di visualizzare il menù contestuale per la modifica di un'associazione.
  Parametri:
  * `e: MouseEvent`
    Parametro contenente i dati dell'evento click sollevato dal browser.
  * `edgeId: String`
    Parametro contenente l'`id` dell'associazione selezionata dall'utente.
- `+ resetNodeEdit(): void`
  Metodo che gestisce l'evento di reset dei campi di modifica di un nodo. Ripristina il `currentNode` presente nello `$scope` utilizzando i dati recuperati da `MindMapService`.
- `- showContextMenu(id: String, functions: Array): void`
  Metodo che visualizza il menù contestuale.
  Parametri:
  * `id: String`
    Parametro che rappresenta l'`id` dell'oggetto sul quale viene visualizzato il menù.
  * `functions: Array`
    Parametro che contiene un `Array` di oggetti rappresentanti le voci del menù. Ogni oggetto ha due campi dati: `name` e `callback`, il primo campo dati specifica il testo associato alla voce del menù e il secondo rappresenta la funzione da chiamare quando l'utente seleziona la voce.
- `+ startAddAssociation(sourceId: String): void`
  Metodo che viene invocato quando l'utente seleziona la voce del menù che permette di aggiungere un'associazione uscente dal nodo selezionato.
  Parametri:
  * `sourceId: String`
    Parametro contenente l'id del nodo sorgente della descrizione.

2.2.2.11 Premi::Front-End::Controllers::NodeContentsEditorController

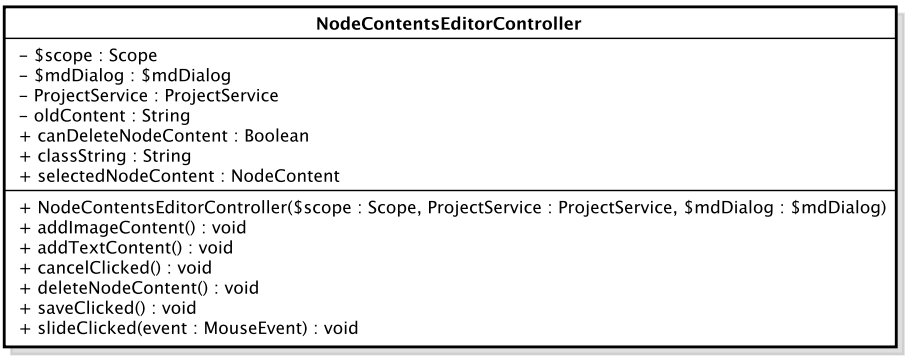


Figura 16: Premi::Front-End::Controllers::NodeContentsEditorController

- **Descrizione**
  Classe che si occupa di gestire la logica legata alla modifica del contenuto di un nodo.
- **Utilizzo**
  Viene utilizzata per consentire all'utente di modificare il contenuto di un nodo.
- **Relazioni con altre classi**:
  - *OUT* `premiNodeContentsEditor`
    Rappresenta il componente grafico che permette all'utente di modificare il contenuto di un nodo. Questo componente fornisce all'utente sia dei campi per inserire/modificare il contenuto del nodo, sia un'anteprima di come questo verrà visualizzato. Dall'anteprima sarà anche possibile spostare e ridimensionare i vari componenti. Saranno presenti inoltre due pulsanti, uno per confermare le modifiche e l'altro per annullarle e ripristinare lo stato del nodo.
  - *OUT* `Node`
    Rappresenta un nodo della mappa mentale. Contiene tutte le informazioni necessarie alla presentazione del contenuto del nodo.
  - *OUT* `NodeContent`
    Rappresenta il contenuto di un nodo. Contiene informazioni che indicano deve essere visualizzato il contenuto all'interno del nodo.
  - *OUT* `ProjectService`
    Questa classe si occupa del recupero e della modifica delle informazioni riguardanti i progetti.
- **Attributi**:
  - `+ canDeleteNodeContent: Boolean`
    Campo dati che specifica se l'oggetto `currentNodeContent` può essere cancellato o meno.
  - `+ classString: String`
    Campo dati contenente le classi CSS dello stile del progetto sotto forma di stringa.
  - `- oldContent: String`
    Campo dati che contiene una copia del contenuto dell'oggetto `NodeContent` che viene selezionato dall'utente. Questo campo dati viene utilizzando per annullare le modifiche dell'utente qualora cercasse di lasciare il contenuto dell'oggetto vuoto.
  - `- ProjectService: ProjectService`
    Campo dati contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.

- `+ selectedNodeContent: NodeContent`
  Campo dati contenente un riferimento all'oggetto NodeContent selezionato dall'utente.
- `- $mdDialog: $mdDialog`
  Campo dati contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
- `- $scope: Scope`
  Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.

- **Metodi**:
  - `+ addImageContent(): void`
    Metodo che aggiunge all'oggetto `node` presente nello `$scope` un nuovo contenuto di tipo immagine.
  - `+ addTextContent(): void`
    Metodo che aggiunge all'oggetto `node` presente nello `$scope` un nuovo contenuto di tipo testuale.
  - `+ cancelClicked(): void`
    Metodo che gestisce il click dell'utente sul pulsante per annullare le modifiche. Questo metodo invoca la funzione `$scope.onReset` e nasconde la directive.
  - `+ deleteNodeContent(): void`
    Metodo che rimuove dall'oggetto `node` presente nello `$scope` l'oggetto `selectedNodeContent`.
  - `+ NodeContentsEditorController($scope: Scope, ProjectService: ProjectService, $mdDialog: $mdDialog)`
    Metodo costruttore della classe.
    Parametri:
    * `$scope: Scope`
      Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
    * `ProjectService: ProjectService`
      Parametro contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
    * `$mdDialog: $mdDialog`
      Parametro contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
  - `+ saveClicked(): void`
    Metodo che gestisce il click dell'utente sul pulsante di conferma delle modifiche. Questo metodo invoca la funzione `$scope.onConfirm` e nasconde la directive.
  - `+ slideClicked(event: MouseEvent): void`
    Metodo che gestisce l'evento `click` sulla slide, deselezionando l'elemento correntemente selezionato.
    Parametri:
    * `event: MouseEvent`
      Parametro contenente le informazioni riguardanti l'evento del browser che ha portato alla chiamata del metodo.

2.2.2.12 Premi::Front-End::Controllers::NodeController

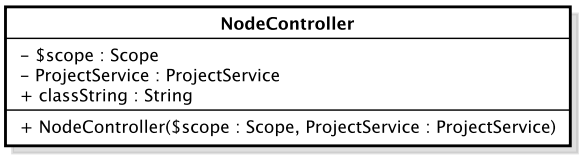


Figura 17: Premi::Front-End::Controllers::NodeController

- **Descrizione**
  Classe che si occupa di gestire il funzionamento della directive `premiNode`.
- **Utilizzo**
  Viene utilizzata per gestire lo stile di visualizzazione del frame che deve essere visualizzato dalla directive.
- **Relazioni con altre classi**:
  - *OUT* `premiNode`
    Rappresenta il componente grafico che visualizza il contenuto di un nodo. Questo contenuto non deve essere modificabile.
  - *OUT* `Node`
    Rappresenta un nodo della mappa mentale. Contiene tutte le informazioni necessarie alla presentazione del contenuto del nodo.
  - *OUT* `ProjectService`
    Questa classe si occupa del recupero e della modifica delle informazioni riguardanti i progetti.
- **Attributi**:
  - `+ classString: String`
    Campo dati contenente le classi CSS che determinano lo stile del progetto.
  - `- ProjectService: ProjectService`
    Campo dati contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
  - `- $scope: Scope`
    Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.
- **Metodi**:
  - `+ NodeController($scope: Scope, ProjectService: ProjectService)`
    Metodo costruttore della classe.
    Viene utilizzato per impostare lo stile di visualizzazione del frame, recuperando le informazioni da `ProjectService`.
    Parametri:
    * `$scope: Scope`
      Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
    * `ProjectService: ProjectService`
      Parametro contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.

2.2.2.13 Premi::Front-End::Controllers::PathController

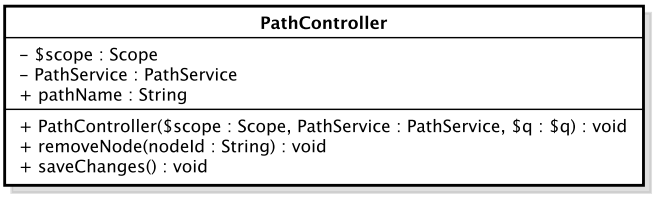


Figura 18: Premi::Front-End::Controllers::PathController

- **Descrizione**
  Classe che gestisce la visibilità e la logica della directive `premiPath`.
- **Utilizzo**
  Viene utilizzata per permettere all'utente di modificare un percorso di presentazione.
- **Relazioni con altre classi**:
  - *OUT* `premiPath`
    Rappresenta il componente grafico che permette all'utente di modificare il percorso selezionato: togliendo dei nodi oppure rinominandolo. Questo componente visualizza la lista dei nodi presenti nel percorso. Per ogni nodo viene visualizzato il titolo del nodo ed un pulsante che permette di rimuoverlo dal percorso.
  - *OUT* `Path`
    Rappresenta un percorso di presentazione costituito da una sequenza ordinata di passi.
  - *OUT* `PathService`
    Questa classe si occupa del recupero e della modifica delle informazioni relative ai percorsi di presentazione relativi al progetto corrente.
- **Attributi**:
  - `+ pathName: String`
    Campo dati contenente il nome del percorso selezionato.
  - `- PathService: PathService`
    Campo dati contenente un riferimento al servizio che si occupa della gestione dei *percorsi*$_G$ di presentazione.
  - `- $scope: Scope`
    Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.
- **Metodi**:
  - `+ PathController($scope: Scope, PathService: PathService)`
    Metodo costruttore della classe.
    Parametri:
    * `$scope: Scope`
      Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
    * `PathService: PathService`
      Parametro contenente un riferimento al servizio che si occupa della gestione dei *percorsi*$_G$ di presentazione.

- `+ removeNode(nodeId: String): void`
  Metodo che rimuove il nodo identificato da `nodeId` dal percorso di presentazione corrente.
  Parametri:
  * `nodeId: String`
    Parametro che contiene l'identificativo del nodo da rimuovere dal percorso.
- `+ saveChanges(): void`
  Metodo che rende effettive le modifiche subite dal percorso di presentazione utilizzando `PathService`.

2.2.2.14 Premi::Front-End::Controllers::PathsEditorController

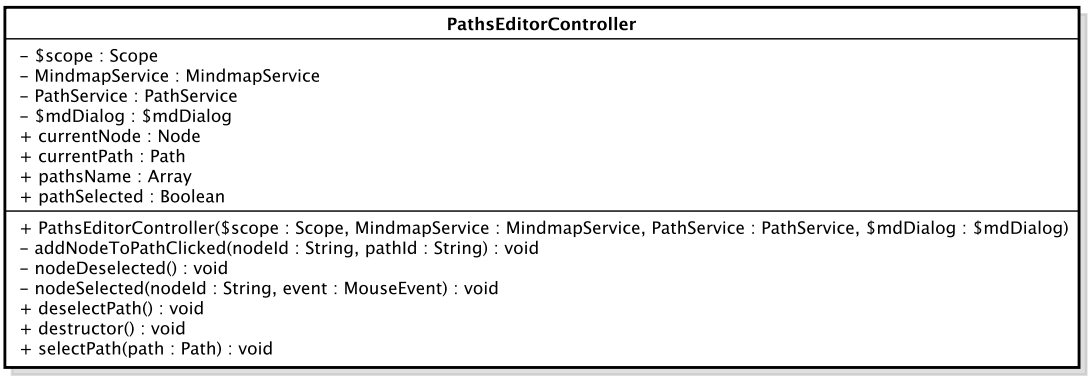


Figura 19: Premi::Front-End::Controllers::PathsEditorController

- **Descrizione**
  Classe che gestisce le operazioni e la logica applicativa riguardante la gestione dei percorsi di presentazione definiti sulla mappa mentale.
- **Utilizzo**
  Questa classe viene utilizzata per consentire all'utente di creare, modificare e cancellare i percorsi di presentazione definiti sulla mappa. In particolare, tramite `PathsEditorView` e le directives che la compongono, permette all'utente di:
  - Creare un nuovo percorso di presentazione;
  - Modificare un percorso di presentazione esistente, rinominandolo oppure aggiungendo/togliendo dei nodi al percorso;
  - Eliminare un percorso di presentazione esistente.

  Tutte queste modifiche vengono effettuate utilizzando `PathService`. Viene fornita anche la possibilità di avviare la presentazione di uno dei percorsi presenti.
- **Relazioni con altre classi**:
  - *OUT* `Node`
    Rappresenta un nodo della mappa mentale. Contiene tutte le informazioni necessarie alla presentazione del contenuto del nodo.
  - *OUT* `Path`
    Rappresenta un percorso di presentazione costituito da una sequenza ordinata di passi.

- *OUT* `MindmapService`
  Questa classe si occupa del recupero e della modifica delle informazioni relative ai nodi e alle associazione presenti nella mappa mentale del progetto corrente. Utilizza `$http` per comunicare con il back-end e `MindmapAdapterService` per memorizzare i dati in locale.
- *OUT* `PathService`
  Questa classe si occupa del recupero e della modifica delle informazioni relative ai percorsi di presentazione relativi al progetto corrente.
- *OUT* `PathsEditorView`
  View dell'applicazione contenente la mappa mentale. Offre funzionalità legate ai percorsi di presentazione. Da questa view è possibile:
  * Creare/eliminare un percorso di presentazione;
  * Modificare un percorso aggiungendo e togliendo nodi, oppure rinominandolo;
  * Scegliere un percorso da presentare e successivamente presentarlo.

- **Attributi**:
  - `+ currentNode: Node`
    Campo dati contenente le informazioni relative al nodo selezionato dall'utente.
  - `+ currentPath: Path`
    Campo dati contenente i dati del percorso selezionato dall'utente, viene utilizzato per memorizzare le modifiche effettuate dall'utente, ma che non sono ancora state confermate.
  - `- MindmapService: MindmapService`
    Campo dati contenente un riferimento al servizio che si occupa della gestione della mappa mentale.
  - `+ pathNames: Array`
    Campo dati contenente le informazioni riguardanti i percorsi di presentazione creati dall'utente. Queste informazioni sono racchiuse in un oggetto con i seguenti attributi: `_id`, `name` e `default`.
  - `+ pathSelected: Boolean`
    Campo dati che specifica se l'utente ha selezionato o meno un percorso di presentazione.
  - `- PathService: PathService`
    Campo dati contenente un riferimento al servizio che si occupa della gestione dei *percorsi*$_G$ di presentazione.
  - `- $mdDialog: $mdDialog`
    Campo dati contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
  - `- $scope: Scope`
    Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.

- **Metodi**:
  - `- addNodeToPathClicked(pathId: String, nodeId: String): void`
    Metodo che gestisce l'evento di aggiunta di un nodo ad un percorso di presentazione. Aggiunge il nodo identificato da `nodeId` in coda al percorso con `id` uguale a `pathId`.
    Parametri:
    * `pathId: String`
      Parametro contenente l'`id` del percorso di presentazione al quale aggiungere il nodo.

    * `nodeId: String`
      Parametro contenente l'id del nodo da aggiungere al percorso di presentazione.

- `+ deselectPath(): void`
  Metodo che deseleziona il percorso di presentazione precedentemente selezionato.
- `- destructor(): void`
  Metodo che gestisce l'evento `destroy` del controller, viene utilizzato per interrompere l'ascolto degli eventi di `MindmapService`.
- `- nodeDeselected(): void`
  Metodo che deseleziona il nodo corrente, impostato l'oggetto `currentNode` a `null`. Questo metodo viene invocato come gestore dell'evento `node-delesect` di `MindmapService`.
- `- nodeSelected(e: MouseEvent, nodeId: String): void`
  Metodo che gestisce l'evento di selezione di un nodo, viene invocato come gestore dell'evento sollevato da `MindmapService`. Si occupa di aggiornare lo `$scope` con i dati del nodo selezionato dall'utente e di rendere visibile la directive per l'aggiunta del nodo selezionato ad un percorso di presentazione.
  Parametri:
    * `e: MouseEvent`
      Parametro contenente i dati dell'evento click sollevato dal browser.
    * `nodeId: String`
      Parametro contenente l'id del nodo selezionato.
- `+ PathsEditorController($scope: Scope, PathService: PathService, MindmapService: MindmapService, $mdDialog: $mdDialog)`
  Metodo costruttore della classe.
  Parametri:
    * `$scope: Scope`
      Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
    * `PathService: PathService`
      Parametro contenente un riferimento al servizio che si occupa della gestione dei *percorsi*$_G$ di presentazione.
    * `MindmapService: MindmapService`
      Parametro contenente un riferimento al servizio che si occupa della gestione della mappa mentale.
    * `$mdDialog: $mdDialog`
      Parametro contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
- `+ selectPath(path: Path): void`
  Metodo che imposta il percorso corrente come riferimento al parametro `path`.
  Parametri:
    * `path: Path`
      Parametro che rappresenta il percorso di presentazione selezionato dall'utente.

2.2.2.15 Premi::Front-End::Controllers::PathsListController

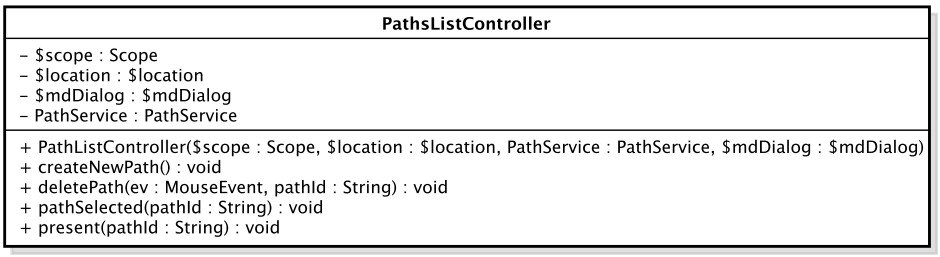


Figura 20: Premi::Front-End::Controllers::PathsListController

- **Descrizione**
  Classe che gestisce le operazioni e la logica applicativa della directive `premiPathsList`.
- **Utilizzo**
  Viene utilizzata per permettere all'utente di interagire con la lista dei percorsi di presentazione presenti nel progetto.
- **Relazioni con altre classi**:
  - *OUT* `premiPathsList`
    Rappresenta il componente grafico che si occupa di visualizzare la lista dei percorsi di presentazione presenti nel progetto corrente. Dalla lista dei percorsi l'utente può selezionare un percorso per modificarlo, cancellare un determinato percorso oppure avviare la presentazione di quel percorso. Questo componente contiene anche un pulsante per creare un nuovo progetto.
  - *OUT* `Path`
    Rappresenta un percorso di presentazione costituito da una sequenza ordinata di passi.
  - *OUT* `PathService`
    Questa classe si occupa del recupero e della modifica delle informazioni relative ai percorsi di presentazione relativi al progetto corrente.
- **Attributi**:
  - `- PathService: PathService`
    Campo dati contenente un riferimento al servizio che si occupa della gestione dei *percorsi*$_G$ di presentazione.
  - `- $location: $location`
    Campo dati contenente un riferimento al servizio creato da *Angular*$_G$ che permette di accedere alla barra degli indirizzi del *browser*$_G$, i cambiamenti all'URL nella barra degli indirizzi si riflettono in questo oggetto e viceversa.
  - `- $mdDialog: $mdDialog`
    Campo dati contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
  - `- $scope: Scope`
    Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.
- **Metodi**:
  - `+ createNewPath(): void`
    Metodo che utilizza `PathService` per creare un nuovo percorso di presentazione.

- `+ deletePath(ev: MouseEvent, pathId: String): void`
  Metodo che richiede a `PathService` la rimozione del percorso identificato da `pathId`.
  Parametri:
  * `ev: MouseEvent`
    Parametro che rappresenta l'evento del browser che ha causato l'invocazione del metodo.
  * `pathId: String`
    Parametro che rappresenta l'identificativo del percorso di presentazione che si vuole cancellare.
- `+ pathSelected(pathId: String): void`
  Metodo che gestisce il click dell'utente sul pulsante di modifica del percorso identificato da `pathId`. Ottiene da `PathService` l'oggetto `Path` rappresentante il percorso di presentazione selezionato ed invoca la funzione `selectPath` presente nello `$scope`.
  Parametri:
  * `pathId: String`
    Parametro che rappresenta l'`id` del percorso che si vuole selezionare.
- `+ PathsListController($scope: Scope, $location: $location, PathService: PathService, $mdDialog: $mdDialog)`
  Metodo costruttore della classe.
  Parametri:
  * `$scope: Scope`
    Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
  * `$location: $location`
    Parametro contenente un riferimento al servizio creato da *Angular*$_G$ che permette di accedere alla barra degli indirizzi del *browser*$_G$, i cambiamenti all'URL nella barra degli indirizzi si riflettono in questo oggetto e viceversa.
  * `PathService: PathService`
    Parametro contenente un riferimento al servizio che si occupa della gestione dei *percorsi*$_G$ di presentazione.
  * `$mdDialog: $mdDialog`
    Parametro contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
- `+ present(pathId: String): void`
  Metodo che avvia la presentazione del percorso identificato da `pathId`.
  Parametri:
  * `pathId: String`
    Parametro che rappresenta l'`id` del percorso che si vuole presentare.

2.2.2.16 Premi::Front-End::Controllers::PresentationController

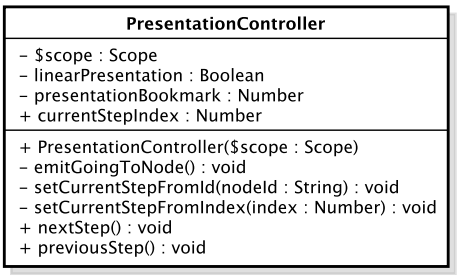


Figura 21: Premi::Front-End::Controllers::PresentationController

- **Descrizione**
  Classe che si occupa di gestire la logica della presentazione di un percorso. La gestione della presentazione funziona mediante l'emissione e l'ascolto di determinati eventi. In particolare questa classe registra dei *listeners* nell'oggetto `$scope` per i seguenti eventi:
  - `presentation-previousStep`: per tornare alla slide precedente;
  - `presentation-nextStep`: per passare alla slide successiva;
  - `presentation-goToId`: per passare alla slide contenente il nodo avente `id` uguale a quello associato all'evento;
  - `presentation-resumeLinear`: per riprendere la presentazione lineare dopo che si è verificato l'evento `presentation-goToId`.

  Viene inoltre sollevato nello `$scope` l'evento `presentation-goingToNode` con l'identificativo del nodo che sta per essere visualizzato ogni volta che si verifica un cambio slide.
- **Utilizzo**
  Viene utilizzata per eseguire la presentazione di un percorso di visualizzazione definito sulla mappa mentale.
- **Relazioni con altre classi**:
  - *OUT* `premiPresentation`
    Rappresenta il componente grafico che genera la presentazione.
- **Attributi**:
  - `+ currentStepIndex: Number`
    Campo dati che rappresenta l'indice della slide corrente, all'interno della sequenza della presentazione.
  - `- linearPresentation: Boolean`
    Campo dati che specifica se la presentazione in corso è lineare o meno, se la presentazione non è lineare i due gestori per gli eventi di passaggio alla slide precedente/successiva sono disattivati.
  - `- presentationBookmark: Number`
    Campo dati contenente l'indice dell'ultima slide del percorso di presentazione visualizzata.
  - `- $scope: Scope`
    Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.
- **Metodi**:

- `- emitGoingToNode(): void`
  Metodo d'utilità che solleva l'evento `presentation-goingToNode` utilizzando i dati del nodo correntemente visualizzato.
- `+ nextStep(): void`
  Metodo che visualizza la slide successiva.
- `+ PresentationController($scope: Scope)`
  Metodo costruttore della classe.
  Parametri:
  * `$scope: Scope`
    Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
- `+ previousStep(): void`
  Metodo che mostra la slide precedente.
- `- setCurrentStepFromId(nodeId: String): void`
  Metodo che visualizza la slide di contenente il nodo avente `id` uguale a `nodeId` presente nella sequenza della presentazione.
  Parametri:
  * `nodeId: String`
    Parametro che rappresenta l'`id` del nodo da visualizzare.
- `- setCurrentStepFromIndex(index: Number): void`
  Metodo che visualizza la slide di indice `index` della sequenza del percorso di presentazione.
  Parametri:
  * `index: Number`
    Parametro contenente l'indice della slide da visualizzare.

2.2.2.17 Premi::Front-End::Controllers::PresentationViewerController

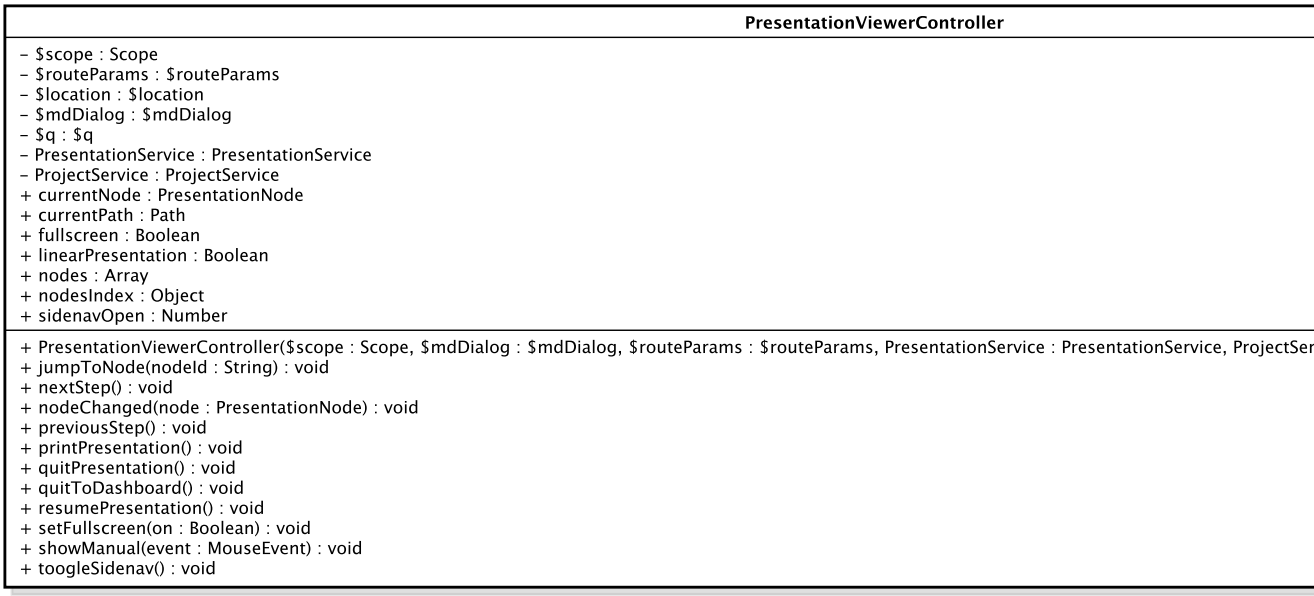


Figura 22: Premi::Front-End::Controllers::PresentationViewerController

- **Descrizione**
  Classe che gestisce le operazioni e la logica applicativa riguardante l'esecuzione di un percorso di presentazione.
- **Utilizzo**
  Viene utilizzata per popolare `PresentationView` con i dati del percorso di presentazione scelto e per gestire gli eventi legati al cambio di frame.
- **Relazioni con altre classi**:
  - *OUT* `Node`
    Rappresenta un nodo della mappa mentale. Contiene tutte le informazioni necessarie alla presentazione del contenuto del nodo.
  - *OUT* `Path`
    Rappresenta un percorso di presentazione costituito da una sequenza ordinata di passi.
  - *OUT* `PresentationNode`
    Classe che estende `Node`, riproducendo una struttura comoda per poter presentare i contenuti di un nodo e semplificare lo spostamento fra frame associati e con relazioni di tipo gerarchico.
  - *OUT* `PathService`
    Questa classe si occupa del recupero e della modifica delle informazioni relative ai percorsi di presentazione relativi al progetto corrente.
  - *OUT* `PresentationService`
    Questa classe si occupa del recupero e della modifica delle informazioni relative alle presentazioni del progetto corrente.
  - *OUT* `PresentationView`
    View dell'applicazione che permette di effettuare la presentazione di un percorso scelto dall'utente. Utilizza la directive `premiPresentation` per offrire funzionalità legate alla presentazione lineare e due menù per permettere all'utente di spostarsi in modo non lineare, seguendo le relazioni definite tra i nodi della mappa mentale. Da questa vista è inoltre possibile stampare ed esportare il percorso di presentazione in esecuzione, mediante le funzionalità offerte dal browser.
- **Attributi**:
  - `+ currentNode: PresentationNode`
    Campo dati contenente le informazioni relative al nodo correntemente visualizzato.
  - `+ currentPath: Path`
    Campo dati contenente le informazioni riguardo al percorso di visualizzazione che viene presentato.
  - `+ fullscreen: Boolean`
    Campo dati che specifica se la presentazione in corso è a schermo intero o meno.
  - `+ linearPresentation: Boolean`
    Campo dati che specifica se la presentazione in corso è lineare o meno.
  - `+ nodes: Array`
    Campo dati contenente una collezione di `PresentationNode` con tutti i nodi presenti nella mappa mentale, ordinati secondo il percorso di presentazione scelto dall'utente.
  - `+ nodesIndex: Object`
    Campo dati contenente le informazioni riguardante la posizione dei nodi nella sequenza della presentazione. Le informazioni all'interno dell'oggetto sono memorizzate come coppie chiave/valore, usando come chiave l'`id` del nodo e come valore un `Array` contenente tutte le posizioni dell'array `nodes` in cui compare quel determinato nodo.

- `- PresentationService: PresentationService`
  Campo dati contenente un riferimento al servizio che si occupa della gestione dei dati relativi alla presentazione.
- `- ProjectService: ProjectService`
  Campo dati contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
- `+ sidenavOpen: Boolean`
  Campo dati che specifica se il menù laterale è aperto o meno.
- `- $location: $location`
  Campo dati contenente un riferimento al servizio creato da *Angular*$_G$ che permette di accedere alla barra degli indirizzi del *browser*$_G$, i cambiamenti all'URL nella barra degli indirizzi si riflettono in questo oggetto e viceversa.
- `- $mdDialog: $mdDialog`
  Campo dati contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
- `- $q: $q`
  Campo dati contenente un riferimento al servizio `$q` creato da *Angular*$_G$ per facilitare la gestione di funzione asincrone mediante l'utilizzo delle promesse.
- `- $routeParams: $routeParams`
  Campo dati contenente un riferimento al servizio creato da *Angular*$_G$ che permette di recuperare i parametri presenti nell'URL di una *view*$_G$.
- `- $scope: Scope`
  Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.

- **Metodi**:
  - `+ jumpToNode(nodeId: String): void`
    Funzione che gestisce il passaggio da un frame ad un frame correlato, permettendo di ottenere una presentazione non lineare.
    Parametri:
    * `nodeId: String`
      Parametro contenente l'id del nodo che sta per essere visualizzato.
  - `+ nextStep(): void`
    Metodo che fa passare la presentazione alla slide successiva, emettendo l'evento `presentation-nextStep`
  - `+ nodeChanged(node: PresentationNode): void`
    Funzione che gestisce l'evento di passaggio da un frame all'altro, si occupa di aggiornare le informazioni riguardanti il nodo correntemente visualizzato.
    Parametri:
    * `node: PresentationNode`
      Parametro contenente il nodo che sta per essere visualizzato.
  - `+ PresentationViewerController($scope: Scope, $mdSidenav: $mdSidenav, $routeParams: $routeParams, PresentationService: PresentationService, ProjectService: ProjectService)`
    Metodo costruttore della classe.
    Parametri:
    * `$scope: Scope`
      Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.

  * `$mdSidenav: $mdSidenav`
    Parametro contenente un riferimento al servizio della libreria *Material for Angular* che permette di gestire le barre di navigazione.
  * `$routeParams: $routeParams`
    Parametro contenente un riferimento al servizio creato da *Angular*$_G$ che permette di recuperare i parametri presenti nell'URL di una *view*$_G$.
  * `PresentationService: PresentationService`
    Parametro contenente un riferimento al servizio che si occupa della gestione dei dati relativi alla presentazione.
  * `ProjectService: ProjectService`
    Parametro contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
- `+ previousStep(): void`
  Metodo che fa passare la presentazione alla slide precedente, emettendo l'evento `presentation-previousStep`
- `+ printPresentation(): void`
  Metodo che si occupa di visualizzare la pagina di stampa per la presentazione in corso.
- `+ quitPresentation(): void`
  Metodo che termina la presentazione in corso e che rimanda l'utente alla pagina per la modifica dei percorsi di presentazione.
- `+ quitToDashboard(): void`
  Metodo che termina la presentazione in corso e che rimanda l'utente alla pagina iniziale.
- `+ resumePresentation(): void`
  Funzione che ripristina la presentazione lineare dopo che l'utente ha navigato ad un frame fuori dalla presentazione.
- `+ setFullscreen(on: Boolean): void`
  Metodo che permette di entrare oppure uscire dalla modalità a schermo intero, se il parametro ricevuto è `true` viene attivata la modalità a schermo interno, altrimenti se è `false` viene disattivata.
  Parametri:
  * `on: Boolean`
    Parametro che specifica se entrare oppure uscire dalla modalità a schermo intero.
- `+ showManual(ev: MouseEvent): void`
  Metodo che visualizza il pop-up contenente il manuale utente.
  Parametri:
  * `ev: MouseEvent`
    Parametro contenente le informazioni relative all'evento del browser che ha portato all'invocazione del metodo.
- `+ toggleSidenav(): void`
  Metodo che inverte il valore del campo dati `sidenavOpen`.

2.2.2.18 Premi::Front-End::Controllers::ProjectSettingsEditorController

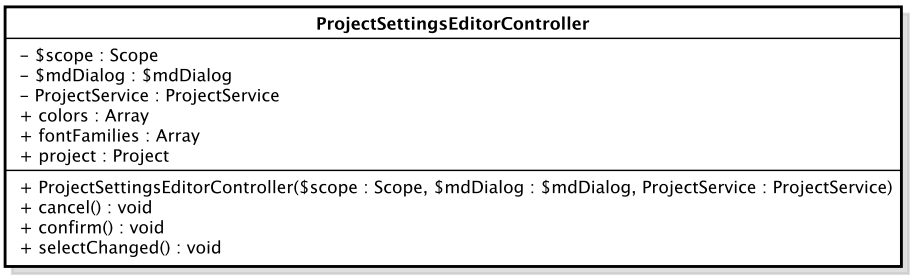


Figura 23: Premi::Front-End::Controllers::ProjectSettingsEditorController

- **Descrizione**
  Classe che si occupa di gestire la logica di funzionamento della directive `premiProjectSettingsEditor`.
- **Utilizzo**
  Viene utilizzata per permettere all'utente di modificare le impostazioni del progetto aperto.
- **Relazioni con altre classi**:
  - *OUT* `premiProjectSettingsEditor`
    Rappresenta il componente grafico che permette di modificare i dati del progetto aperto. Questo componente presenta i vari campi per interagire con i parametri del progetto e due pulsanti che l'utente utilizza per confermare o annullare le modifiche.
  - *OUT* `Project`
    Rappresenta un progetto creato dall'utente. Contiene le informazioni riguardanti i parametri globali del progetto: nome, formato del testo, colore di sfondo.
  - *OUT* `ProjectService`
    Questa classe si occupa del recupero e della modifica delle informazioni riguardanti i progetti.
- **Attributi**:
  - `+ colors: Array`
    Campo dati contente una lista di tutti i colori disponibili per il progetto.
  - `+ fontFamilies: Array`
    Campo dati contente una lista di tutti i font disponibili per il progetto.
  - `+ project: Project`
    Campo dati contenente un oggetto che viene utilizzato per memorizzare le modifiche effettuate dall'utente sulle impostazioni del progetto e che non sono ancora state confermate. L'oggetto ha i seguenti campi dati: `name`, `textColor`, `backgroundColor` e `fontFamily`.
  - `- ProjectService: ProjectService`
    Campo dati contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
  - `- $mdDialog: $mdDialog`
    Campo dati contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
  - `- $scope: Scope`
    Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.
- **Metodi**:

- `+ cancel(): void`
  Metodo che gestisce l'evento di click sul pulsante che nasconde la directive.
- `+ confirm(): void`
  Metodo che viene invocato quando l'utente conferma le modifiche al progetto, si occupa di salvare le modifiche utilizzando `ProjectService`.
- `+ ProjectSettingsEditorController($scope: Scope, $mdDialog: $mdDialog, ProjectService: ProjectService)`
  Metodo costruttore della classe.
  Parametri:
  * `$scope: Scope`
    Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
  * `$mdDialog: $mdDialog`
    Parametro contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
  * `ProjectService: ProjectService`
    Parametro contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
- `+ selectChanged(): void`
  Metodo che aggiorna l'anteprima dello stile del progetto, viene invocato ogni volta che l'utente seleziona una nuova opzione per lo stile del progetto.

2.2.2.19 Premi::Front-End::Controllers::ProjectsListController

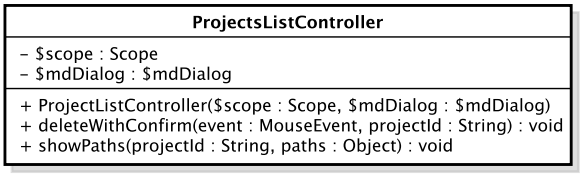


Figura 24: Premi::Front-End::Controllers::ProjectsListController

- **Descrizione**
  Classe che gestisce la logica di funzionamento della directive `premiProjectsList`.
- **Utilizzo**
  Viene utilizzata per gestire il popup della directive che permette di scegliere un percorso da presentare.
- **Relazioni con altre classi**:
  - *OUT* `premiProjectsList`
    Rappresenta il componente grafico che mostra la lista dei progetti creati dall'utente. Per ogni progetto della lista è presente un pulsante che permette di modificarlo, di cancellarlo e di presentarlo.
- **Attributi**:
  - `- $mdDialog: $mdDialog`
    Campo dati contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.

- `- $scope: Scope`
  Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.

- **Metodi**:
  - `+ deleteWithConfirm(event: MouseEvent, projectId: String): void`
    Metodo che gestisce la cancellazione di un progetto dalla lista dei progetti. Prima di effettuare la cancellazione richiede all'utente la conferma dell'operazione.
    Parametri:
    * `event: MouseEvent`
      Parametro che rappresenta l'evento del browser che ha causato l'invocazione del metodo.
    * `projectId: String`
      Parametro che rappresenta l'`id` del progetto da cancellare.
  - `+ ProjectsListController($scope: Scope, $mdDialog: $mdDialog)`
    Metodo costruttore della classe.
    Parametri:
    * `$scope: Scope`
      Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
    * `$mdDialog: $mdDialog`
      Parametro contenente un riferimento al servizio della libreria *Material for Angular* che permette di creare delle componenti a popup.
  - `+ showPaths(projectId: String, paths: Object): void`
    Metodo che rende visibile la finestra di dialogo per la scelta del percorso di presentazione da presentare.
    Parametri:
    * `projectId: String`
      Parametro rappresentate l'`id` del progetto contenente il percorso da presentare.
    * `paths: Object`
      Parametro rappresentante i percorsi disponibili, contiene le informazioni sotto forma di coppie chiave/valore, come chiave viene utilizzato l'`id` e come valore il nome.

2.2.2.20 Premi::Front-End::Controllers::RegistrationController

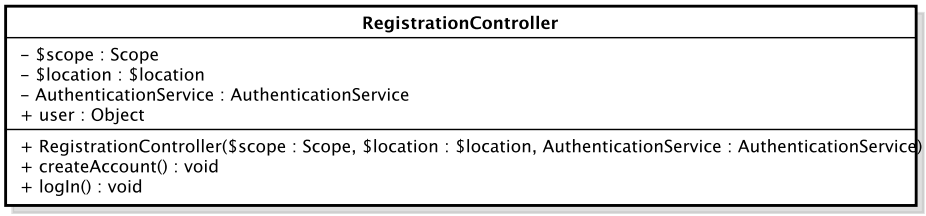


Figura 25: Premi::Front-End::Controllers::RegistrationController

- **Descrizione**
  Classe che gestisce le operazioni e la logica applicativa riguardante la registrazione di un utente.
- **Utilizzo**
  Viene utilizzata per gestire la view di registrazione e per registrare un account sul server, sfruttando `AuthenticationService`. Prima di proporre all'utente la view di registrazione controlla che non ci sia una sessione già attiva, in tal caso reindirizza l'utente alla lista dei progetti disponibili.
- **Relazioni con altre classi**:
  - *OUT* `AuthenticationService`
    Questa classe si occupa di gestire il processo di autenticazione e di registrazione di un utente.
  - *OUT* `RegistrationView`
    View dell'applicazione contenente il form per la registrazione di un nuovo utente. Contiene un link alla pagina di autenticazione.
- **Attributi**:
  - `- AuthenticationService: AuthenticationService`
    Campo dati contenente un riferimento al servizio che si occupa della gestione delle informazioni legate all'autenticazione.
  - `+ user: Object`
    Campo dati contenente un oggetto con i seguenti attributi: `email`, `password` e `passwordCheck`.
  - `- $location: $location`
    Campo dati contenente un riferimento al servizio creato da *Angular*$_G$ che permette di accedere alla barra degli indirizzi del *browser*$_G$, i cambiamenti all'URL nella barra degli indirizzi si riflettono in questo oggetto e viceversa.
  - `- $scope: Scope`
    Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.
- **Metodi**:
  - `+ createAccount(): void`
    Metodo che gestisce l'evento di pressione del pulsante per la registrazione. Legge dallo `$scope` i valori inseriti dall'utente e prova a registrare un account sul server, utilizzando `AuthenticationService`. Se la registrazione va a buon fine, effettua in modo automatico il login e reindirizza l'utente alla `DashboardView`, altrimenti mostra un messaggio d'errore che spiega perché non è stato possibile eseguire la registrazione.
  - `+ logIn(): void`
    Metodo che gestisce l'evento click sul pulsante di login. Effettua il redirect alla pagina di login.
  - `+ RegistrationController($scope: Scope, $location: $location, AuthenticationService: AuthenticationService)`
    Metodo costruttore della classe.
    Parametri:
    * `$scope: Scope`
      Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.

* `$location: $location`
  Parametro contenente un riferimento al servizio creato da *Angular*$_G$ che permette di accedere alla barra degli indirizzi del *browser*$_G$, i cambiamenti all'URL nella barra degli indirizzi si riflettono in questo oggetto e viceversa.
* `AuthenticationService: AuthenticationService`
  Parametro contenente un riferimento al servizio che si occupa della gestione delle informazioni legate all'autenticazione.

2.2.2.21 Premi::Front-End::Controllers::SmartMenuController

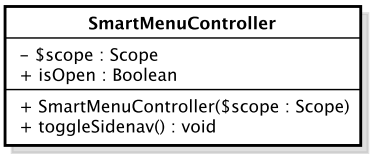


Figura 26: Premi::Front-End::Controllers::SmartMenuController

- **Descrizione**
  Classe che gestisce la visibilità della directive `premiSmartMenu`.
- **Utilizzo**
  Viene utilizzata per permettere all'utente di nascondere il menù con i nodi associati al nodo visualizzato in presentazione.
- **Relazioni con altre classi**:
  - *OUT* `premiSmartMenu`
    Rappresenta il menù che durante la presentazione permette di navigare la mappa mentale seguendo le associazioni definite tra i vari nodi. Questo componente consiste in una lista di elementi selezionabili dall'utente, ogni elemento corrisponde ad un'associazione presente tra il nodo corrente ed un altro nodo della mappa.
- **Attributi**:
  - `+ isOpen: Boolean`
    Campo dati che specifica se il menù è aperto o chiuso.
  - `- $scope: Scope`
    Campo dati contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$. Contiene gli oggetti che definiscono il model dell'applicazione.
- **Metodi**:
  - `+ SmartMenuController($scope: Scope)`
    Metodo costruttore della classe.
    Parametri:
    * `$scope: Scope`
      Parametro contenente un riferimento all'oggetto `$scope` creato da *Angular*$_G$, viene utilizzato come mezzo di comunicazione tra il *controller*$_G$ e la *view*$_G$ e contiene gli oggetti che definiscono il model dell'applicazione.
  - `+ toggleSidenav(): void`
    Metodo che inverte il valore del campo dati `isOpen`.

2.3 Premi::Front-End::Directives

2.3.1 Informazioni generali

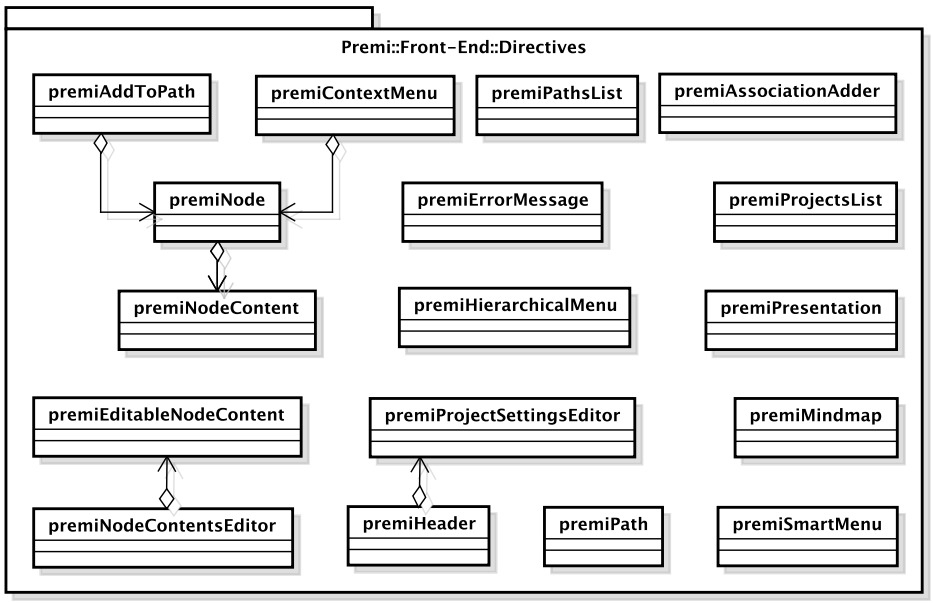


Figura 27: Premi::Front-End::Directives

- **Descrizione**
  Package contenente le directives che compongno le views.
- **Padre**: `Front-End`
- **Interazioni con altri componenti**:
  - `Controllers`
    Package contenente i controller della componente front-end dell'applicazione.
  - `Model`
    Package contenente le classi che definiscono la business logic dell'applicazione.
  - `Views`
    Package contenente le views della componente front-end dell'applicazione.

2.3.2 Classi

2.3.2.1 Premi::Front-End::Directives::premiAddToPath

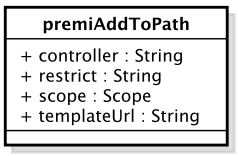


Figura 28: Premi::Front-End::Directives::premiAddToPath

- **Descrizione**
  Rappresenta il componente grafico che permette all'utente di visualizzare il contenuto di un nodo e di aggiungerlo ad uno dei percorsi di presentazione esistenti. Questo componente visualizza una lista contenente tutti i percorsi disponibili al quale, selezionando quello desiderato, è possibile aggiungere il nodo slezionato.
- **Utilizzo**
  Viene utilizzato per consentire all'utente di aggiungere un nodo ad un percorso di presentazione.
- **Relazioni con altre classi**:
  - *IN* `AddToPathController`
    Classe che si occupa di gestire la logica di funzionamento della directive `premiAddToPath`.
  - *IN* `PathsEditorView`
    View dell'applicazione contenente la mappa mentale. Offre funzionalità legate ai percorsi di presentazione. Da questa view è possibile:
    * Creare/eliminare un percorso di presentazione;
    * Modificare un percorso aggiungendo e togliendo nodi, oppure rinominandolo;
    * Scegliere un percorso da presentare e successivamente presentarlo.
  - *OUT* `premiNode`
    Rappresenta il componente grafico che visualizza il contenuto di un nodo. Questo contenuto non deve essere modificabile.
  - *OUT* `Node`
    Rappresenta un nodo della mappa mentale. Contiene tutte le informazioni necessarie alla presentazione del contenuto del nodo.
- **Attributi**:
  - `+ controller: String`
    Stringa contenente il nome del controller della directive.
  - `+ restrict: String`
    Stringa che specifica le modalità che permettono di inserire una directive all'interno della pagina.
  - `+ scope: Scope`
    Oggetto scope della directive. Questo scope è locale alla directive ed è situato al di fuori della gerarchia principale della pagina.
    Lo scope deve contenere i seguenti oggetti:
    * `node (=)`: oggetto di tipo `Node` contenente i dati del nodo che si vuole aggiungere al percorso;
    * `paths (=)`: oggetto contenente le informazioni riguardanti i percorsi presenti all'interno del progetto. Queste informazioni sono memorizzate come coppie chiave/valore, usando l'`id` del percorso come chiave e il nome come valore;
    * `onAdd (&)`: funzione da invocare quanto l'utente conferma l'inserimento di un nodo in un percorso. La funzione viene invocata con due parametri di tipo stringa rappresentanti l'id del nodo e l'id del percorso in cui inserirlo.
  - `+ templateUrl: String`
    Stringa contenente il *percorso*$_G$ dove è situato il file *HTML*$_G$ che contiene il *template*$_G$ della directive.

2.3.2.2 Premi::Front-End::Directives::premiAssociationAdder

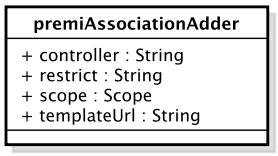


Figura 29: Premi::Front-End::Directives::premiAssociationAdder

- **Descrizione**
  Rappresenta il componente grafico che permette all'utente di creare un'associazione tra il nodo selezionato e un altro nodo della mappa.
- **Utilizzo**
  Viene utilizzata per consentire all'utente di aggiungere un'associazione tra il nodo selezionato ed un altro nodo della mappa.
- **Relazioni con altre classi**:
  - *IN* `AssociationAdderController`
    Classe che gestisce il funzionamento della directive `premiAssociationAdder`.
  - *IN* `MindmapEditorView`
    View dell'applicazione contenente la mappa mentale. Offre funzionalità di modifica della mappa. Da questa view è possibile:
    * Aggiungere/togliere nodi;
    * Modificare il contenuto di un nodo;
    * Aggiungere/togliere associazioni tra nodi;
    * Modificare i parametri del progetto.
- **Attributi**:
  - `+ controller: String`
    Stringa contenente il nome del controller della directive.
  - `+ restrict: String`
    Stringa che specifica le modalità che permettono di inserire una directive all'interno della pagina.
  - `+ scope: Scope`
    Oggetto scope della directive. Questo scope è locale alla directive ed è situato al di fuori della gerarchia principale della pagina.
    Lo scope deve contenere i seguenti oggetti:
    * `nodes (=)`: oggetto contenente le informazioni riguardanti i nodi con i quali è possibile definire un'associazione, le informazioni sono memorizzate come coppie chiave/valore, usando come chiave l'textttid del nodo e come valore il titolo;
    * `onNodeSelected (&)`: funzione da invocare quando l'utente seleziona un nodo tra quelli disponibili, la funzione viene invocata con un parametro di tipo stringa corrispondente all'id del nodo selezionato.
  - `+ templateUrl: String`
    Stringa contenente il *percorso*$_G$ dove è situato il file *HTML*$_G$ che contiene il *template*$_G$ della directive.

2.3.2.3 Premi::Front-End::Directives::premiContextMenu

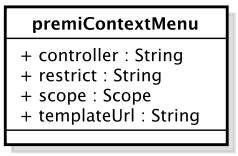


Figura 30: Premi::Front-End::Directives::premiContextMenu

- **Descrizione**
  Rappresenta un menù contestuale generato in base agli oggetti passati nello scope isolato. Fornisce un pulsante per ogni oggetto ricevuto come parametro, ogni pulsante viene rappresentato con un'icona e con del testo. Al click di un pulsante viene invocata la funzione ad esso associata.
- **Utilizzo**
  Viene utilizzato per realizzare i menù che vengono visualizzati quando l'utente seleziona un nodo o una relazione presente nella mappa.
- **Relazioni con altre classi**:
  - *IN* `ContextMenuController`
    Classe che gestisce il comportamento della directive `premiContextMenu`.
  - *IN* `MindmapEditorView`
    View dell'applicazione contenente la mappa mentale. Offre funzionalità di modifica della mappa. Da questa view è possibile:
    * Aggiungere/togliere nodi;
    * Modificare il contenuto di un nodo;
    * Aggiungere/togliere associazioni tra nodi;
    * Modificare i parametri del progetto.
  - *OUT* `premiNode`
    Rappresenta il componente grafico che visualizza il contenuto di un nodo. Questo contenuto non deve essere modificabile.
- **Attributi**:
  - `+ controller: String`
    Stringa contenente il nome del controller della directive.
  - `+ restrict: String`
    Stringa che specifica le modalità che permettono di inserire una directive all'interno della pagina.
  - `+ scope: Scope`
    Oggetto scope della directive. Questo scope è locale alla directive ed è situato al di fuori della gerarchia principale della pagina.
    Lo scope deve contenere i seguenti oggetti:
    * `objectId (@)`: stringa contenente l'id dell'oggetto sul quale viene visualizzato il menù;
    * `node (=)`: (opzionale) oggetto di tipo `Node` del quale visualizzare l'anteprima;
    * `operations (=)`: `Array` di oggetti contenenti i seguenti campi dati:
      · `name`: stringa contenente il nome da visualizzare;
      · `callback`: funzione da eseguire quando l'utente seleziona la voce del menù, viene invocata con `objectId` come parametro;
      · `image`: stringa contenente l'URL dell'immagine da visualizzare di fianco alla voce del menù.

- `+ templateUrl: String`
  Stringa contenente il *percorso*$_G$ dove è situato il file *HTML*$_G$ che contiene il *template*$_G$ della directive.

2.3.2.4 Premi::Front-End::Directives::premiEditableNodeContent

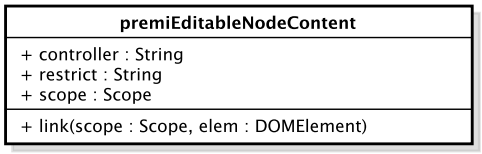


Figura 31: Premi::Front-End::Directives::premiEditableNodeContent

- **Descrizione**
  Rappresenta un elemento contenuto nel frame di un nodo. Questo elemento è spostabile e ridimensionabile. Le modifiche effettuate su questo elemento vengono riportate direttamente sull'oggetto `nodeContent`. Questo componente deve essere in grado di rappresentare le varie tipologie di contenuto che possono essere presenti all'interno di un nodo. La discriminazione del tipo del contenuto deve essere fatta in base ai campi dati dell'oggetto `nodeContent`.
- **Utilizzo**
  Viene utilizzato per permettere all'utente di spostare un elemento contenuto all'interno di un frame.
- **Relazioni con altre classi**:
  - *IN* `EditableNodeContentController`
    Classe che gestisce la logica della directive `premiEditableNodeContent`. Questa classe registra un gestore per l'evento `nodecontent-deselect` che imposta come de-selezionata la directive.
  - *IN* `premiNodeContentsEditor`
    Rappresenta il componente grafico che permette all'utente di modificare il contenuto di un nodo. Questo componente fornisce all'utente sia dei campi per inserire/modificare il contenuto del nodo, sia un'anteprima di come questo verrà visualizzato. Dall'anteprima sarà anche possibile spostare e ridimensionare i vari componenti. Saranno presenti inoltre due pulsanti, uno per confermare le modifiche e l'altro per annullarle e ripristinare lo stato del nodo.
  - *OUT* `NodeContent`
    Rappresenta il contenuto di un nodo. Contiene informazioni che indicano deve essere visualizzato il contenuto all'interno del nodo.
- **Attributi**:
  - `+ controller: String`
    Stringa contenente il nome del controller della directive.
  - `+ restrict: String`
    Stringa che specifica le modalità che permettono di inserire una directive all'interno della pagina.
  - `+ scope: Scope`
    Oggetto scope della directive. Questo scope è locale alla directive ed è situato al di fuori della gerarchia principale della pagina.
    Dentro questo oggetto devono essere presenti i seguenti oggetti:

    * `nodeContent (=)`: oggetto che rappresenta un elemento contenuto nel frame di un nodo;
    * `container (=)`: oggetto contenente le informazioni riguardanti le dimensioni del contenitore della directive, in particolare l'oggetto ha due campi dati: `width` e `height`.

- **Metodi**:
  - `+ link(scope: Scope, elem: DOMElement)`
    Funzione che viene invocata da *Angular*$_G$ durante la fase di compilazione della pagina. Permette manipolare il DOM, aggiungendo nuovi elementi o definendo un comportamento per l'elemento.
    Viene utilizzata per registrare i vari gestori degli eventi del mouse in modo da poter effettuare il *drag'n'drop* e vengono aggiunti ulteriori elementi al DOM per poter effettuare il ridimensionamento dell'elemento.
    Parametri:
    * `scope: Scope`
      Parametro che contiene un riferimento all'oggetto `scope` della directive.
    * `elem: DOMElement`
      Parametro contenente un oggetto della libreria jQuery che rappresenta l'oggetto del DOM sul quale è definita la directive.

2.3.2.5 Premi::Front-End::Directives::premiErrorMessage

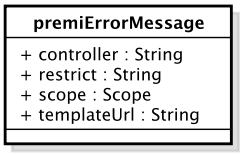


Figura 32: Premi::Front-End::Directives::premiErrorMessage

- **Descrizione**
  Rappresenta il componente grafico che permette di mostrare messaggi d'errore all'utente all'interno dell'applicazione. Questo componente fornisce anche un pulsante che permette di nascondere il messaggio.
- **Utilizzo**
  Viene utilizzato per comunicare all'utente gli errori relativi ai dati inseriti o ad operazioni non consentite che l'utente cerca di effetturare.
- **Relazioni con altre classi**:
  - *IN* `AppRun`
    Classe che si occupa di gestire l'inizializzazione dell'applicazione.
  - *IN* `ErrorMessageController`
    Classe che gestisce la logica di visualizzazione della directive `premiErrorMessage`.
  - *OUT* `ErrorInfo`
    Rappresenta le informazioni di un errore che si è verificato eseguendo una determinata operazione.
- **Attributi**:

- `+ controller: String`
  Stringa contenente il nome del controller della directive.
- `+ restrict: String`
  Stringa che specifica le modalità che permettono di inserire una directive all'interno della pagina.
- `+ scope: Scope`
  Oggetto scope della directive. Questo scope è locale alla directive ed è situato al di fuori della gerarchia principale della pagina.
  Lo scope riceve un oggetto di tipo `ErrorInfo` contenente tutte le informazioni riguardanti l'errore che si è verificato.
- `+ templateUrl: String`
  Stringa contenente il $percorso_G$ dove è situato il file $HTML_G$ che contiene il $template_G$ della directive.

2.3.2.6 Premi::Front-End::Directives::premiHeader

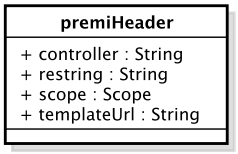


Figura 33: Premi::Front-End::Directives::premiHeader

- **Descrizione**
  Rappresenta la barra di navigazione che permette all'utente di spostarsi tra le varie views dell'applicazione. Fornisce inoltre i pulsanti per visualizzare il manuale utente, effettuare il logout, chiudere al presentazione e il progetto. Inoltre, quando l'utente si trova nelle views in cui è possibile modificare il progetto, questa directive permette di mostrare la finestra di modifica delle impostazioni del progetto, sia di interagire con la visualizzazione della mappa mentale, modificando lo zoom e permettendo anche di stamparla.
- **Utilizzo**
  Viene utilizzata per consentire all'utente di spostarsi tra la varie views dell'applicazione, per effettuare il logout e per accedere al manuale dell'applicazione.
- **Relazioni con altre classi**:
  - *IN* `HeaderController`
    Classe che gestisce le operazioni e la logica applicativa della directive `premiHeader`,
  - *IN* `MindmapEditorView`
    View dell'applicazione contenente la mappa mentale. Offre funzionalità di modifica della mappa. Da questa view è possibile:
    * Aggiungere/togliere nodi;
    * Modificare il contenuto di un nodo;
    * Aggiungere/togliere associazioni tra nodi;
    * Modificare i parametri del progetto.
  - *IN* `PathsEditorView`
    View dell'applicazione contenente la mappa mentale. Offre funzionalità legate ai percorsi di presentazione. Da questa view è possibile:
    * Creare/eliminare un percorso di presentazione;

    * Modificare un percorso aggiungendo e togliendo nodi, oppure rinominandolo;
    * Scegliere un percorso da presentare e successivamente presentarlo.
  - *OUT* `premiProjectSettingsEditor`
    Rappresenta il componente grafico che permette di modificare i dati del progetto aperto. Questo componente presenta i vari campi per interagire con i parametri del progetto e due pulsanti che l'utente utilizza per confermare o annullare le modifiche.

- **Attributi**:
  - `+ controller: String`
    Stringa contenente il nome del controller della directive.
  - `+ restrict: String`
    Stringa che specifica le modalità che permettono di inserire una directive all'interno della pagina.
  - `+ scope: Scope`
    Oggetto scope della directive. Questo scope è locale alla directive ed è situato al di fuori della gerarchia principale della pagina.
  - `+ templateUrl: String`
    Stringa contenente il *percorso*$_G$ dove è situato il file *HTML*$_G$ che contiene il *template*$_G$ della directive.

2.3.2.7 Premi::Front-End::Directives::premiHierarchicalMenu

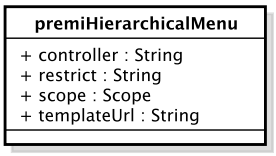


Figura 34: Premi::Front-End::Directives::premiHierarchicalMenu

- **Descrizione**
  Rappresenta il menù che durante la presentazione permette di navigare la mappa mentale seguendo le relazioni gerarchiche presenti tra i vari nodi della mappa mentale. Questo componente consiste in una lista di elementi selezionabili dall'utente, ogni elemento corrisponde ad una relazione presente tra il nodo corrente ed un altro nodo della mappa.
- **Utilizzo**
  Viene utilizzata per permettere all'utente di spostarsi tra i nodi della mappa mentale seguendo le relazioni gerarchiche presenti.
- **Relazioni con altre classi**:
  - *IN* `HierarchicalMenuController`
    Classe che gestisce la visibilità della directive `premiHierarchicalMenu`.
  - *IN* `PresentationView`
    View dell'applicazione che permette di effettuare la presentazione di un percorso scelto dall'utente. Utilizza la directive `premiPresentation` per offrire funzionalità legate alla presentazione lineare e due menù per permettere all'utente di spostarsi in modo non lineare, seguendo le relazioni definite tra i nodi della mappa mentale. Da questa vista è inoltre possibile stampare ed esportare il percorso di presentazione in esecuzione, mediante le funzionalità offerte dal browser.

- *OUT* `NodeReference`
  Rappresenta il riferimento ad un nodo della mappa mentale, contiene l'identificativo e il titolo del del nodo.

- **Attributi**:
  - `+ controller: String`
    Stringa contenente il nome del controller della directive.
  - `+ restrict: String`
    Stringa che specifica le modalità che permettono di inserire una directive all'interno della pagina.
  - `+ scope: Scope`
    Oggetto scope della directive. Questo scope è locale alla directive ed è situato al di fuori della gerarchia principale della pagina.
    Lo scope deve contenere i seguenti oggetti:
    * `parent (=)`: oggetto di tipo `NodeReference` contenente un riferimento al nodo padre del nodo correntemente visualizzato;
    * `relations (=): Array` contenente oggetti di tipo `NodeReference`, rappresentanti le associazioni che coinvolgono il nodo correntemente visualizzato;
    * `onClick (&)`: funzione da invocare quando l'utente clicca su un elemento del menù. La funzione viene invocata con un parametro corrispondete all'id del nodo che compare nella relazione selezionata dall'utente.
  - `+ templateUrl: String`
    Stringa contenente il $percorso_G$ dove è situato il file $HTML_G$ che contiene il $template_G$ della directive.

2.3.2.8 Premi::Front-End::Directives::premiMindmap

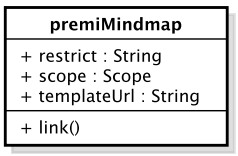


Figura 35: Premi::Front-End::Directives::premiMindmap

- **Descrizione**
  Rappresenta il componente grafico che disegna la mappa mentale.
- **Utilizzo**
  Viene utilizzato per inserire all'interno delle varie view dell'applicazione la mappa mentale.
- **Relazioni con altre classi**:
  - *IN* `MindmapService`
    Questa classe si occupa del recupero e della modifica delle informazioni relative ai nodi e alle associazione presenti nella mappa mentale del progetto corrente. Utilizza `$http` per comunicare con il back-end e `MindmapAdapterService` per memorizzare i dati in locale.
  - *IN* `PathsEditorView`
    View dell'applicazione contenente la mappa mentale. Offre funzionalità legate ai percorsi di presentazione. Da questa view è possibile:

    * Creare/eliminare un percorso di presentazione;
    * Modificare un percorso aggiungendo e togliendo nodi, oppure rinominandolo;
    * Scegliere un percorso da presentare e successivamente presentarlo.

- **Attributi**:
  - `+ replace: Boolean`
    Valore booleano che specifica se il template della directive andrà a sostituire o meno l'elemento del DOM nel quale la directive è dichiarata.
  - `+ restrict: String`
    Stringa che specifica le modalità che permettono di inserire una directive all'interno della pagina.
  - `+ scope: Scope`
    Oggetto scope della directive. Questo scope è locale alla directive ed è situato al di fuori della gerarchia principale della pagina.
  - `+ templateUrl: String`
    Stringa contenente il *percorso*$_G$ dove è situato il file *HTML*$_G$ che contiene il *template*$_G$ della directive.
- **Metodi**:
  - `+ link()`
    Funzione che viene invocata da *Angular*$_G$ durante la fase di compilazione della pagina. Permette manipolare il DOM, aggiungendo nuovi elementi o definendo un comportamento per l'elemento.
    Viene utilizzata per disegnare la mappa mentale.

2.3.2.9 Premi::Front-End::Directives::premiNode

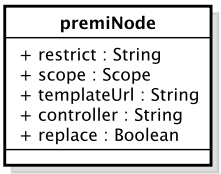


Figura 36: Premi::Front-End::Directives::premiNode

- **Descrizione**
  Rappresenta il componente grafico che visualizza il contenuto di un nodo. Questo contenuto non deve essere modificabile.
- **Utilizzo**
  Viene utilizzato per visualizzare il contenuto di un nodo.
- **Relazioni con altre classi**:
  - *IN* `NodeController`
    Classe che si occupa di gestire il funzionamento della directive `premiNode`.
  - *IN* `premiAddToPath`
    Rappresenta il componente grafico che permette all'utente di visualizzare il contenuto di un nodo e di aggiungerlo ad uno dei percorsi di presentazione esistenti. Questo componente visualizza una lista contenente tutti i percorsi disponibili al quale, selezionando quello desiderato, è possibile aggiungere il nodo slezionato.

- *IN* `premiContextMenu`
  Rappresenta un menù contestuale generato in base agli oggetti passati nello scope isolato. Fornisce un pulsante per ogni oggetto ricevuto come parametro, ogni pulsante viene rappresentato con un'icona e con del testo. Al click di un pulsante viene invocata la funzione ad esso associata.
- *OUT* `premiNodeContent`
  Rappresenta un elemento contenuto nel frame di un nodo. Questo componente deve essere in grado di rappresentare le varie tipologie di contenuto che possono essere presenti all'interno di un nodo. La discriminazione del tipo del contenuto deve essere fatta in base ai campi dati dell'oggetto `nodeContent`.
- *OUT* `Node`
  Rappresenta un nodo della mappa mentale. Contiene tutte le informazioni necessarie alla presentazione del contenuto del nodo.
- *OUT* `PresentationNode`
  Classe che estende `Node`, riproducendo una struttura comoda per poter presentare i contenuti di un nodo e semplificare lo spostamento fra frame associati e con relazioni di tipo gerarchico.

- **Attributi**:
  - `- controller: String`
    Stringa contenente il nome del controller della directive.
  - `+ replace: Boolean`
    Valore booleano che specifica se il template della directive andrà a sostituire o meno l'elemento del DOM nel quale la directive è dichiarata.
  - `+ restrict: String`
    Stringa che specifica le modalità che permettono di inserire una directive all'interno della pagina.
  - `+ scope: Scope`
    Oggetto scope della directive. Questo scope è locale alla directive ed è situato al di fuori della gerarchia principale della pagina.
    Dentro questo oggetto devono essere presenti i seguenti oggetti:
    * `node (=)`: oggetto di tipo `Node` che deve essere visualizzato.
  - `+ templateUrl: String`
    Stringa contenente il $percorso_G$ dove è situato il file $HTML_G$ che contiene il $template_G$ della directive.

2.3.2.10 Premi::Front-End::Directives::premiNodeContent

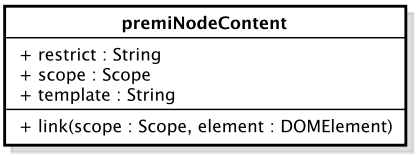


Figura 37: Premi::Front-End::Directives::premiNodeContent

- **Descrizione**
  Rappresenta un elemento contenuto nel frame di un nodo. Questo componente deve essere in grado di rappresentare le varie tipologie di contenuto che possono essere presenti all'interno di un

nodo. La discriminazione del tipo del contenuto deve essere fatta in base ai campi dati dell'oggetto `nodeContent`.

- **Utilizzo**
  Viene utilizzata per rappresentare un elemento all'interno del frame di un nodo.
- **Relazioni con altre classi**:
  - *IN* `premiNode`
    Rappresenta il componente grafico che visualizza il contenuto di un nodo. Questo contenuto non deve essere modificabile.
  - *OUT* `NodeContent`
    Rappresenta il contenuto di un nodo. Contiene informazioni che indicano deve essere visualizzato il contenuto all'interno del nodo.
- **Attributi**:
  - `+ restrict: String`
    Stringa che specifica le modalità che permettono di inserire una directive all'interno della pagina.
  - `+ scope: Scope`
    Oggetto scope della directive. Questo scope è locale alla directive ed è situato al di fuori della gerarchia principale della pagina.
    Dentro questo oggetto deve essere presente il seguente oggetto:
    * `nodeContent (=)`: oggetto di tipo `NodeContent` che rappresenta un elemento contenuto nel frame di un nodo.
  - `+ template: String`
    Stringa che definisce il *template*$_G$ *HTML*$_G$ della directive.
- **Metodi**:
  - `+ link(scope: Scope, element: DOMElement)`
    Funzione che viene invocata da *Angular*$_G$ durante la fase di compilazione della pagina. Permette manipolare il DOM, aggiungendo nuovi elementi o definendo un comportamento per l'elemento.
    Viene utilizzata per creare l'elemento del DOM corretto in base al tipo di contenuto dell'oggetto e per gestire l'evento `resize` della pagina.
    Parametri:
    * `scope: Scope`
      Parametro contenente un riferimento allo `scope` della directive.
    * `element: DOMElement`
      Parametro contenente un oggetto della libreria jQuery rappresentante l'elemento del DOM contenente la directive.

2.3.2.11 Premi::Front-End::Directives::premiNodeContentsEditor

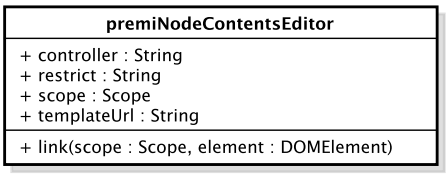


Figura 38: Premi::Front-End::Directives::premiNodeContentsEditor

- **Descrizione**
  Rappresenta il componente grafico che permette all'utente di modificare il contenuto di un nodo. Questo componente fornisce all'utente sia dei campi per inserire/modificare il contenuto del nodo, sia un'anteprima di come questo verrà visualizzato. Dall'anteprima sarà anche possibile spostare e ridimensionare i vari componenti. Saranno presenti inoltre due pulsanti, uno per confermare le modifiche e l'altro per annullarle e ripristinare lo stato del nodo.
- **Utilizzo**
  Viene utilizzata per permettere all'utente di modificare i nodi della mappa menale.
- **Relazioni con altre classi**:
  - *IN* `NodeContentsEditorController`
    Classe che si occupa di gestire la logica legata alla modifica del contenuto di un nodo.
  - *IN* `MindmapEditorView`
    View dell'applicazione contenente la mappa mentale. Offre funzionalità di modifica della mappa. Da questa view è possibile:
    * Aggiungere/togliere nodi;
    * Modificare il contenuto di un nodo;
    * Aggiungere/togliere associazioni tra nodi;
    * Modificare i parametri del progetto.
  - *OUT* `premiEditableNodeContent`
    Rappresenta un elemento contenuto nel frame di un nodo. Questo elemento è spostabile e ridimensionabile. Le modifiche effettuate su questo elemento vengono riportate direttamente sull'oggetto `nodeContent`. Questo componente deve essere in grado di rappresentare le varie tipologie di contenuto che possono essere presenti all'interno di un nodo. La discriminazione del tipo del contenuto deve essere fatta in base ai campi dati dell'oggetto `nodeContent`.
  - *OUT* `Node`
    Rappresenta un nodo della mappa mentale. Contiene tutte le informazioni necessarie alla presentazione del contenuto del nodo.
- **Attributi**:
  - `+ controller: String`
    Stringa contenente il nome del controller della directive.
  - `+ restrict: String`
    Stringa che specifica le modalità che permettono di inserire una directive all'interno della pagina.
  - `+ scope: Scope`
    Oggetto scope della directive. Questo scope è locale alla directive ed è situato al di fuori della gerarchia principale della pagina.
    All'interno dello scope dovranno essere presenti i seguenti oggetti:
    * `node (=)`: oggetto di tipo nodo che deve essere modificato;

    * `onConfirm (&)`: funzione invocata quando l'utente conferma le modifiche effettuate;
    * `onReset (&)`: funzione invocata quando l'utente decide di annullare le modifiche.
  - `+ templateUrl: String`
    Stringa contenente il *percorso*$_G$ dove è situato il file *HTML*$_G$ che contiene il *template*$_G$ della directive.

- **Metodi**:
  - `+ link(scope: Scope, element: DOMElement)`
    Funzione che viene invocata da *Angular*$_G$ durante la fase di compilazione della pagina. Permette manipolare il DOM, aggiungendo nuovi elementi o definendo un comportamento per l'elemento.
    Viene utilizzata per registrare un gestore per l'evento `resize` del browser, in modo da poter scalare in modo corretto i contenuti del nodo.
    Parametri:
    * `scope: Scope`
      Parametro contenente un riferimento allo `scope` della directive.
    * `element: DOMElement`
      Parametro contenente un oggetto della libreria jQuery rappresentante l'elemento del DOM contenente la directive.

2.3.2.12 Premi::Front-End::Directives::premiPath

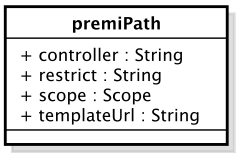


Figura 39: Premi::Front-End::Directives::premiPath

- **Descrizione**
  Rappresenta il componente grafico che permette all'utente di modificare il percorso selezionato: togliendo dei nodi oppure rinominandolo. Questo componente visualizza la lista dei nodi presenti nel percorso. Per ogni nodo viene visualizzato il titolo del nodo ed un pulsante che permette di rimuoverlo dal percorso.
- **Utilizzo**
  Viene utilizzata per consentire all'utente di modificare i percorsi di presentazione che ha creato.
- **Relazioni con altre classi**:
  - *IN* `PathController`
    Classe che gestisce la visibilità e la logica della directive `premiPath`.
  - *IN* `PathsEditorView`
    View dell'applicazione contenente la mappa mentale. Offre funzionalità legate ai percorsi di presentazione. Da questa view è possibile:
    * Creare/eliminare un percorso di presentazione;
    * Modificare un percorso aggiungendo e togliendo nodi, oppure rinominandolo;
    * Scegliere un percorso da presentare e successivamente presentarlo.

- *OUT* `NodeReference`
  Rappresenta il riferimento ad un nodo della mappa mentale, contiene l'identificativo e il titolo del del nodo.
- *OUT* `Path`
  Rappresenta un percorso di presentazione costituito da una sequenza ordinata di passi.

- **Attributi**:
  - `+ controller: String`
    Stringa contenente il nome del controller della directive.
  - `+ restrict: String`
    Stringa che specifica le modalità che permettono di inserire una directive all'interno della pagina.
  - `+ scope: Scope`
    Oggetto scope della directive. Questo scope è locale alla directive ed è situato al di fuori della gerarchia principale della pagina.
    Lo scope deve contenere i seguenti oggetti:
    * `selectedPath (=)`: oggetto `Path` da visualizzare e modificare;
    * `deselectPath (&)`: funzione da invocare quando viene nascosta la directive.
  - `+ templateUrl: String`
    Stringa contenente il $percorso_G$ dove è situato il file $HTML_G$ che contiene il $template_G$ della directive.

2.3.2.13 Premi::Front-End::Directives::premiPathsList

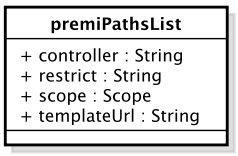


Figura 40: Premi::Front-End::Directives::premiPathsList

- **Descrizione**
  Rappresenta il componente grafico che si occupa di visualizzare la lista dei percorsi di presentazione presenti nel progetto corrente. Dalla lista dei percorsi l'utente può selezionare un percorso per modificarlo, cancellare un determinato percorso oppure avviare la presentazione di quel percorso. Questo componente contiene anche un pulsante per creare un nuovo progetto.
- **Utilizzo**
  Viene utilizzata per mostrare all'utente i percorsi già definiti nel progetto corrente e per creare un percorso vuoto.
- **Relazioni con altre classi**:
  - *IN* `PathsListController`
    Classe che gestisce le operazioni e la logica applicativa della directive `premiPathsList`.
  - *IN* `PathsEditorView`
    View dell'applicazione contenente la mappa mentale. Offre funzionalità legate ai percorsi di presentazione. Da questa view è possibile:
    * Creare/eliminare un percorso di presentazione;

    * Modificare un percorso aggiungendo e togliendo nodi, oppure rinominandolo;
    * Scegliere un percorso da presentare e successivamente presentarlo.

- **Attributi**:
  - `+ controller: String`
    Stringa contenente il nome del controller della directive.
  - `+ restrict: String`
    Stringa che specifica le modalità che permettono di inserire una directive all'interno della pagina.
  - `+ scope: Scope`
    Oggetto scope della directive. Questo scope è locale alla directive ed è situato al di fuori della gerarchia principale della pagina.
    Lo scope dovrà contenere i seguenti oggetti:
    * `paths (=): Array` contenente oggetti definiti con il JSON ricevuto in risposta dalle API del server, in particolare questi oggetti hanno i seguenti campi dati: `_id`, `name` e `default`;
    * `selectPath (&)` funzione da invocare quando l'utente preme il pulsante di modifica su un percorso presente in lista.
  - `+ templateUrl: String`
    Stringa contenente il $percorso_G$ dove è situato il file $HTML_G$ che contiene il $template_G$ della directive.

2.3.2.14 Premi::Front-End::Directives::premiPresentation

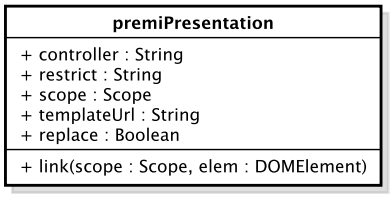


Figura 41: Premi::Front-End::Directives::premiPresentation

- **Descrizione**
  Rappresenta il componente grafico che genera la presentazione.
- **Utilizzo**
  Viene utilizzata per generare uno slideshow a partire dai nodi presenti nella mappa mentale.
- **Relazioni con altre classi**:
  - *IN* `PresentationController`
    Classe che si occupa di gestire la logica della presentazione di un percorso. La gestione della presentazione funziona mediante l'emissione e l'ascolto di determinati eventi. In particolare questa classe registra dei *listeners* nell'oggetto `$scope` per i seguenti eventi:
    * `presentation-previousStep`: per tornare alla slide precedente;
    * `presentation-nextStep`: per passare alla slide successiva;
    * `presentation-goToId`: per passare alla slide contenente il nodo avente `id` uguale a quello associato all'evento;
    * `presentation-resumeLinear`: per riprendere la presentazione lineare dopo che si è verificato l'evento `presentation-goToId`.

Viene inoltre sollevato nello `$scope` l'evento `presentation-goingToNode` con l'identificativo del nodo che sta per essere visualizzato ogni volta che si verifica un cambio slide.

- *IN* `PresentationView`
  View dell'applicazione che permette di effettuare la presentazione di un percorso scelto dall'utente. Utilizza la directive `premiPresentation` per offrire funzionalità legate alla presentazione lineare e due menù per permettere all'utente di spostarsi in modo non lineare, seguendo le relazioni definite tra i nodi della mappa mentale. Da questa vista è inoltre possibile stampare ed esportare il percorso di presentazione in esecuzione, mediante le funzionalità offerte dal browser.

- **Attributi**:
  - `+ controller: String`
    Stringa contenente il nome del controller della directive.
  - `+ replace: Boolean`
    Valore booleano che specifica se il template della directive andrà a sostituire o meno l'elemento del DOM nel quale la directive è dichiarata.
  - `+ restrict: String`
    Stringa che specifica le modalità che permettono di inserire una directive all'interno della pagina.
  - `+ scope: Scope`
    Oggetto scope della directive. Questo scope è locale alla directive ed è situato al di fuori della gerarchia principale della pagina.
    Lo scope deve contenere i seguenti oggetti:
    * `paths (=)`: oggetto di tipo `Path` contenente le informazioni riguardanti il percorso che si vuole presentare;
    * `nodes (=)`: `Array` contenente gli oggetti di tipo `PresentationNode` con i quali generare la presentazione;
    * `nodesIndex (=)`: oggetto contenente le informazioni riguardante la posizione dei nodi nella sequenza della presentazione. Le informazioni all'interno dell'oggetto sono memorizzate come coppie chiave/valore, usando come chiave l'id del nodo e come valore un `Array` contenente tutte le posizioni dell'array `nodes` in cui compare quel determinato nodo.
  - `+ templateUrl: String`
    Stringa contenente il *percorso*$_G$ dove è situato il file *HTML*$_G$ che contiene il *template*$_G$ della directive.

- **Metodi**:
  - `+ link(scope: Scope, elem: DOMElement)`
    Funzione che viene invocata da *Angular*$_G$ durante la fase di compilazione della pagina. Permette manipolare il DOM, aggiungendo nuovi elementi o definendo un comportamento per l'elemento.
    Viene utilizzata per integrare le funzionalità di Impress con Angular.
    Parametri:
    * `scope: Scope`
      Parametro contenente un riferimento allo `scope` della directive.
    * `elem: DOMElement`
      Parametro contenente un oggetto della libreria *jQuery* che rappresenta l'elemento del DOM associato alla directive.

2.3.2.15 Premi::Front-End::Directives::premiProjectSettingsEditor

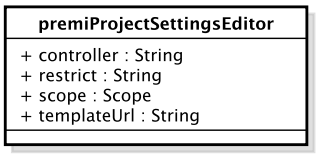


Figura 42: Premi::Front-End::Directives::premiProjectSettingsEditor

- **Descrizione**
  Rappresenta il componente grafico che permette di modificare i dati del progetto aperto. Questo componente presenta i vari campi per interagire con i parametri del progetto e due pulsanti che l'utente utilizza per confermare o annullare le modifiche.
- **Utilizzo**
  Fornisce all'utente i controlli necessari per modificare i parametri del progetto correntemente aperto.
- **Relazioni con altre classi**:
  - *IN* `ProjectSettingsEditorController`
    Classe che si occupa di gestire la logica di funzionamento della directive `premiProjectSettingsEditor`.
  - *IN* `premiHeader`
    Rappresenta la barra di navigazione che permette all'utente di spostarsi tra le varie views dell'applicazione. Fornisce inoltre i pulsanti per visualizzare il manuale utente, effettuare il logout, chiudere al presentazione e il progetto. Inoltre, quando l'utente si trova nelle views in cui è possibile modificare il progetto, questa directive permette di mostrare la finestra di modifica delle impostazioni del progetto, sia di interagire con la visualizzazione della mappa mentale, modificando lo zoom e permettendo anche di stamparla.
  - *OUT* `Project`
    Rappresenta un progetto creato dall'utente. Contiene le informazioni riguardanti i parametri globali del progetto: nome, formato del testo, colore di sfondo.
- **Attributi**:
  - `+ controller: String`
    Stringa contenente il nome del controller della directive.
  - `+ restrict: String`
    Stringa che specifica le modalità che permettono di inserire una directive all'interno della pagina.
  - `+ scope: Scope`
    Oggetto scope della directive. Questo scope è locale alla directive ed è situato al di fuori della gerarchia principale della pagina.
  - `+ templateUrl: String`
    Stringa contenente il $percorso_G$ dove è situato il file $HTML_G$ che contiene il $template_G$ della directive.

2.3.2.16 Premi::Front-End::Directives::premiProjectsList

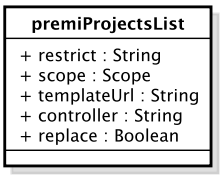


Figura 43: Premi::Front-End::Directives::premiProjectsList

- **Descrizione**
  Rappresenta il componente grafico che mostra la lista dei progetti creati dall'utente. Per ogni progetto della lista è presente un pulsante che permette di modificarlo, di cancellarlo e di presentarlo.
- **Utilizzo**
  Viene utilizzata per permettere all'utente di interagire con i progetti che ha creato.
- **Relazioni con altre classi**:
  - *IN* `ProjectsListController`
    Classe che gestisce la logica di funzionamento della directive `premiProjectsList`.
  - *IN* `DashboardView`
    View dell'applicazione contenente la lista dei progetti dell'utente correntemente autenticato. Da questa view è possibile:
    * Creare un nuovo progetto;
    * Aprire un progetto esistente per modificarlo oppure presentarlo;
    * Cancellare un progetto esistente.
  - *OUT* `Project`
    Rappresenta un progetto creato dall'utente. Contiene le informazioni riguardanti i parametri globali del progetto: nome, formato del testo, colore di sfondo.
- **Attributi**:
  - `+ controller: String`
    Stringa contenente il nome del controller della directive.
  - `+ replace: Boolean`
    Valore booleano che specifica se il template della directive andrà a sostituire o meno l'elemento del DOM nel quale la directive è dichiarata.
  - `+ restrict: String`
    Stringa che specifica le modalità che permettono di inserire una directive all'interno della pagina.
  - `+ scope: Scope`
    Oggetto scope della directive. Questo scope è locale alla directive ed è situato al di fuori della gerarchia principale della pagina.
    Lo scope deve contenere i seguenti oggetti:
    * `projects (=)`: array contenente i progetti da visualizzare;
    * `onEdit (&)`: funzione da invocare quando l'utente seleziona un progetto da aprire/modificare, la funzione viene invocata con un parametro corrispondente all'id del progetto;
    * `onDelete (&)`: funzione da invocare quando l'utente seleziona un progetto da eliminare, la funzione viene invocata con un parametro corrispondente all'id del progetto;

  * `onPlay (&)`: funzione da invocare quando l'utente ha scelto un percorso di presentazione da eseguire, la funzione viene invocata con due parametri di tipo stringa, il primo rappresentante l'id del progetto e il secondo rappresentante l'id del percorso da presentare.
- `+ templateUrl: String`
  Stringa contenente il $percorso_G$ dove è situato il file $HTML_G$ che contiene il $template_G$ della directive.

2.3.2.17 Premi::Front-End::Directives::premiSmartMenu

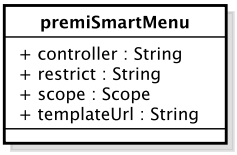


Figura 44: Premi::Front-End::Directives::premiSmartMenu

- **Descrizione**
  Rappresenta il menù che durante la presentazione permette di navigare la mappa mentale seguendo le associazioni definite tra i vari nodi. Questo componente consiste in una lista di elementi selezionabili dall'utente, ogni elemento corrisponde ad un'associazione presente tra il nodo corrente ed un altro nodo della mappa.
- **Utilizzo**
  Viene utilizzata per permettere all'utente di spostarsi tra i nodi della mappa mentale seguendo le assocazioni presenti.
- **Relazioni con altre classi**:
  - *IN* `SmartMenuController`
    Classe che gestisce la visibilità della directive `premiSmartMenu`.
  - *IN* `PresentationView`
    View dell'applicazione che permette di effettuare la presentazione di un percorso scelto dall'utente. Utilizza la directive `premiPresentation` per offrire funzionalità legate alla presentazione lineare e due menù per permettere all'utente di spostarsi in modo non lineare, seguendo le relazioni definite tra i nodi della mappa mentale. Da questa vista è inoltre possibile stampare ed esportare il percorso di presentazione in esecuzione, mediante le funzionalità offerte dal browser.
  - *OUT* `NodeReference`
    Rappresenta il riferimento ad un nodo della mappa mentale, contiene l'identificativo e il titolo del del nodo.
- **Attributi**:
  - `+ controller: String`
    Stringa contenente il nome del controller della directive.
  - `+ restrict: String`
    Stringa che specifica le modalità che permettono di inserire una directive all'interno della pagina.
  - `+ scope: Scope`
    Oggetto scope della directive. Questo scope è locale alla directive ed è situato al di fuori della gerarchia principale della pagina.
    Lo scope deve contenere i seguenti oggetti:

    * `relations (=): Array` contenente oggetti di tipo `NodeReference`, rappresentanti le associazioni che coinvolgono il nodo correntemente visualizzato;
    * `onClick (&)`: funzione da invocare quando l'utente clicca su un elemento del menù. La funzione viene invocata con un parametro corrispondete all'id del nodo che compare nella relazione selezionata dall'utente.
  - `+ templateUrl: String`
    Stringa contenente il $percorso_G$ dove è situato il file $HTML_G$ che contiene il $template_G$ della directive.

2.4 Premi::Front-End::Model

2.4.1 Informazioni generali

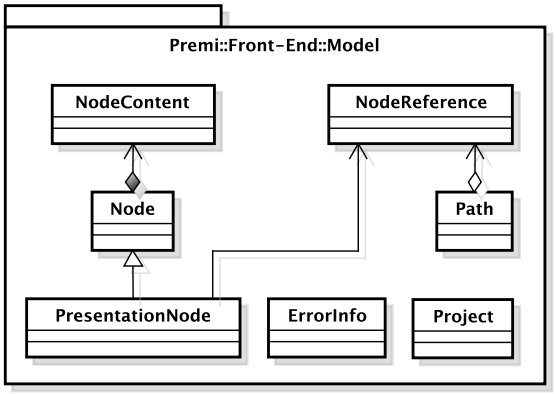


Figura 45: Premi::Front-End::Model

- **Descrizione**
  Package contenente le classi che definiscono la business logic dell'applicazione.
- **Padre**: `Front-End`
- **Interazioni con altri componenti**:
  - `Controllers`
    Package contenente i controller della componente front-end dell'applicazione.
  - `Directives`
    Package contenente le directives che compongno le views.
  - `Services`
    Package contenente le classi che descrivono il meccanismo con cui il front-end può interfacciarsi con le API REST del back-end. Permette di popolare il model del client e di richiedere l'esecuzione di operazioni al server.
  - `Views`
    Package contenente le views della componente front-end dell'applicazione.

2.4.2 Classi

2.4.2.1 Premi::Front-End::Model::ErrorInfo

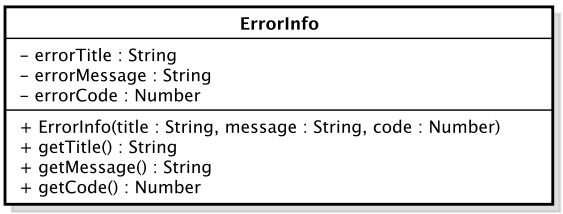


Figura 46: Premi::Front-End::Model::ErrorInfo

- **Descrizione**
  Rappresenta le informazioni di un errore che si è verificato eseguendo una determinata operazione.
- **Utilizzo**
  Viene utilizzata per racchiudere tutte le informazioni riguardanti l'errore.
- **Relazioni con altre classi**:
  - *IN* `AppRun`
    Classe che si occupa di gestire l'inizializzazione dell'applicazione.
  - *IN* `premiErrorMessage`
    Rappresenta il componente grafico che permette di mostrare messaggi d'errore all'utente all'interno dell'applicazione. Questo componente fornisce anche un pulsante che permette di nascondere il messaggio.
  - *IN* `AuthenticationService`
    Questa classe si occupa di gestire il processo di autenticazione e di registrazione di un utente.
  - *IN* `MindmapService`
    Questa classe si occupa del recupero e della modifica delle informazioni relative ai nodi e alle associazione presenti nella mappa mentale del progetto corrente. Utilizza `$http` per comunicare con il back-end e `MindmapAdapterService` per memorizzare i dati in locale.
  - *IN* `PathService`
    Questa classe si occupa del recupero e della modifica delle informazioni relative ai percorsi di presentazione relativi al progetto corrente.
  - *IN* `PresentationService`
    Questa classe si occupa del recupero e della modifica delle informazioni relative alle presentazioni del progetto corrente.
  - *IN* `ProjectService`
    Questa classe si occupa del recupero e della modifica delle informazioni riguardanti i progetti.
- **Attributi**:
  - `- errorCode: Number`
    Campo dati che contiene il codice dell'errore che si è verificato.
  - `- errorMessage: String`
    Campo dati contenente la descrizione dell'errore.
  - `- errorTitle: String`
    Campo dati che rappresenta il titolo del messaggio d'errore.
- **Metodi**:

- `+ ErrorInfo(title: String, message: String, code: Number)`
  Metodo costruttore della classe.
  Parametri:
  * `title: String`
    Parametro contenente il titolo del messaggio d'errore.
  * `message: String`
    Parametro contenente al descrizione del messaggio d'errore.
  * `code: Number`
    Parametro contenente il codice del messaggio d'errore.
- `+ getCode(): Number`
  Metodo *getter* per il campo dati `errorCode`.
- `+ getMessage(): String`
  Metodo *getter* per il campo dati `errorMessage`.
- `+ getTitle(): String`
  Metodo *getter* per il campo dati `errorTitle`.

2.4.2.2 Premi::Front-End::Model::Node

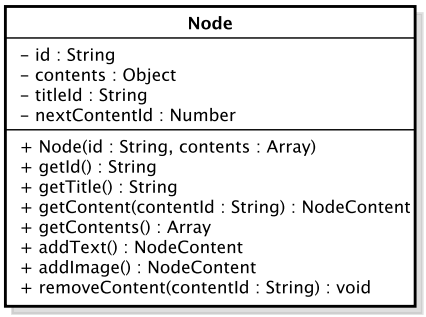


Figura 47: Premi::Front-End::Model::Node

- **Descrizione**
  Rappresenta un nodo della mappa mentale. Contiene tutte le informazioni necessarie alla presentazione del contenuto del nodo.
- **Utilizzo**
  Viene utilizzata per memorizzare i dati di un nodo.
- **Sottoclassi**:
  - `PresentationNode`
- **Relazioni con altre classi**:
  - *IN* `ContextMenuController`
    Classe che gestisce il comportamento della directive `premiContextMenu`.
  - *IN* `MindmapEditorController`
    Classe che gestisce le operazioni e la logica applicativa riguardante la modifica di una mappa.
  - *IN* `NodeContentsEditorController`
    Classe che si occupa di gestire la logica legata alla modifica del contenuto di un nodo.

- *IN* `NodeController`
  Classe che si occupa di gestire il funzionamento della directive `premiNode`.
- *IN* `PathsEditorController`
  Classe che gestisce le operazioni e la logica applicativa riguardante la gestione dei percorsi di presentazione definiti sulla mappa mentale.
- *IN* `PresentationViewerController`
  Classe che gestisce le operazioni e la logica applicativa riguardante l'esecuzione di un percorso di presentazione.
- *IN* `premiAddToPath`
  Rappresenta il componente grafico che permette all'utente di visualizzare il contenuto di un nodo e di aggiungerlo ad uno dei percorsi di presentazione esistenti. Questo componente visualizza una lista contenente tutti i percorsi disponibili al quale, selezionando quello desiderato, è possibile aggiungere il nodo slezionato.
- *IN* `premiNode`
  Rappresenta il componente grafico che visualizza il contenuto di un nodo. Questo contenuto non deve essere modificabile.
- *IN* `premiNodeContentsEditor`
  Rappresenta il componente grafico che permette all'utente di modificare il contenuto di un nodo. Questo componente fornisce all'utente sia dei campi per inserire/modificare il contenuto del nodo, sia un'anteprima di come questo verrà visualizzato. Dall'anteprima sarà anche possibile spostare e ridimensionare i vari componenti. Saranno presenti inoltre due pulsanti, uno per confermare le modifiche e l'altro per annullarle e ripristinare lo stato del nodo.
- *IN* `MindmapService`
  Questa classe si occupa del recupero e della modifica delle informazioni relative ai nodi e alle associazione presenti nella mappa mentale del progetto corrente. Utilizza `$http` per comunicare con il back-end e `MindmapAdapterService` per memorizzare i dati in locale.
- *IN* `MindmapEditorView`
  View dell'applicazione contenente la mappa mentale. Offre funzionalità di modifica della mappa. Da questa view è possibile:
  * Aggiungere/togliere nodi;
  * Modificare il contenuto di un nodo;
  * Aggiungere/togliere associazioni tra nodi;
  * Modificare i parametri del progetto.
- *IN* `PathsEditorView`
  View dell'applicazione contenente la mappa mentale. Offre funzionalità legate ai percorsi di presentazione. Da questa view è possibile:
  * Creare/eliminare un percorso di presentazione;
  * Modificare un percorso aggiungendo e togliendo nodi, oppure rinominandolo;
  * Scegliere un percorso da presentare e successivamente presentarlo.
- *OUT* `NodeContent`
  Rappresenta il contenuto di un nodo. Contiene informazioni che indicano deve essere visualizzato il contenuto all'interno del nodo.

- **Attributi**:
  - `- contents: Object`
    Attributo che rappresenta i dati contenuti nel nodo, i vari oggetti `NodeContent` vengono memorizzati all'interno dell'oggetto come coppie chiave/valore, usando come chiavi gli `id` devi vari `NodeContent`.

- `- id: String`
  Attributo che rappresenta l'identificativo del nodo.
- `- nextContentId: Number`
  Campo dati contenente l'`id` da dare al prossimo contenuto che verrà aggiunto al nodo.
- `- titleId: String`
  Attributo che rappresenta l'id del `NodeContent` presente nell'array `content` e che contiene il titolo del nodo.

- **Metodi**:
  - `+ addImage(): NodeContent`
    Metodo che aggiunge ai contenuti del nodo un'immagine utilizzando dei dati di default. Viene ritornato un riferimento all'oggetto creato.
  - `+ addText(): NodeContent`
    Metodo che aggiunge ai contenuti del nodo un elemento testuale utilizzando dei dati di default. Viene ritornato un riferimento all'oggetto creato.
  - `+ getContent(contentId: String): NodeContent`
    Metodo che ritorna l'oggetto di `NodeContent` presente all'interno del nodo avente `id` uguale a quello ricevuto come parametro.
    Parametri:
    * `contentId: String`
      Parametro che rappresenta l'`id` dell'oggetto richiesto.
  - `+ getContents(): Array`
    Metodo che restituisce tutti gli oggetti `NodeContent` contenuti nel nodo.
  - `+ getId(): String`
    Metodo che ritorna l'`id` del nodo.
  - `+ getTitle(): String`
    Metodo che, uttilizza il campo dati `titleId` per recuperare dall'array `contents`, restituisce una stringa contenente il titolo del nodo.
  - `+ Node(id: String, contents: Array)`
    Metodo costruttore della classe.
    Parametri:
    * `id: String`
      Parametro che rappresenta l'`id` da dare al nuovo oggetto.
    * `contents: Array`
      Parametro che rappresenta i contenuti da inserire all'interno del nuovo nodo. L'array deve contenere almeno un `NodeContent` di tipo 'title'. All'interno dell'array gli oggetti possono essere di tipo `NodeContent` oppure essere definiti con il `JSON` ricevuto in risposta dalle API del back-end.
  - `+ removeContent(contentId: String): void`
    Metodo che rimuove dalla collezione `content` l'elemento avente `id` uguale a `contentId`.
    Parametri:
    * `contentId: String`
      Parametro che rappresenta l'`id` dell'oggetto `NodeContent` da rimuovere.

2.4.2.3 Premi::Front-End::Model::NodeContent

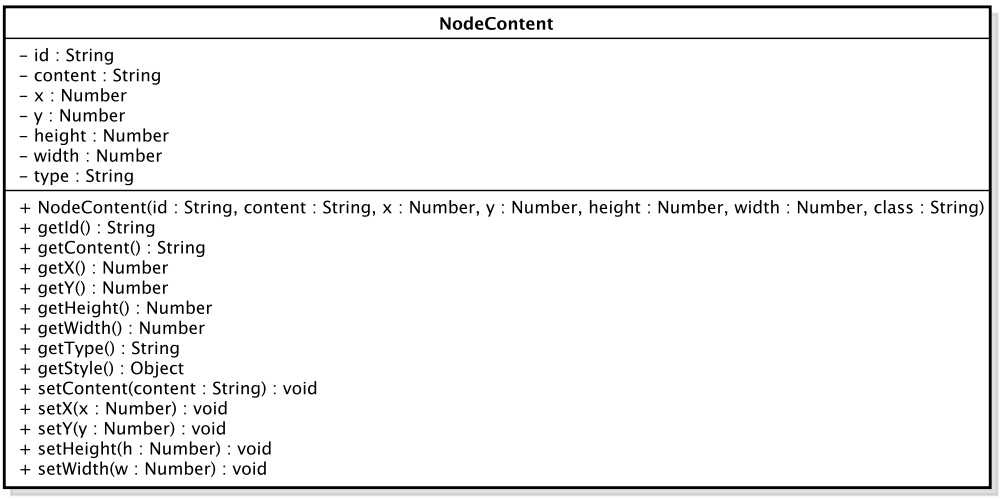


Figura 48: Premi::Front-End::Model::NodeContent

- **Descrizione**
  Rappresenta il contenuto di un nodo. Contiene informazioni che indicano deve essere visualizzato il contenuto all'interno del nodo.
- **Utilizzo**
  Viene utilizzata per memorizzare i vari oggetti che sono contenuti in un nodo.
- **Relazioni con altre classi**:
  - *IN* `NodeContentsEditorController`
    Classe che si occupa di gestire la logica legata alla modifica del contenuto di un nodo.
  - *IN* `premiEditableNodeContent`
    Rappresenta un elemento contenuto nel frame di un nodo. Questo elemento è spostabile e ridimensionabile. Le modifiche effettuate su questo elemento vengono riportate direttamente sull'oggetto `nodeContent`. Questo componente deve essere in grado di rappresentare le varie tipologie di contenuto che possono essere presenti all'interno di un nodo. La discriminazione del tipo del contenuto deve essere fatta in base ai campi dati dell'oggetto `nodeContent`.
  - *IN* `premiNodeContent`
    Rappresenta un elemento contenuto nel frame di un nodo. Questo componente deve essere in grado di rappresentare le varie tipologie di contenuto che possono essere presenti all'interno di un nodo. La discriminazione del tipo del contenuto deve essere fatta in base ai campi dati dell'oggetto `nodeContent`.
  - *IN* `Node`
    Rappresenta un nodo della mappa mentale. Contiene tutte le informazioni necessarie alla presentazione del contenuto del nodo.
  - *IN* `PresentationNode`
    Classe che estende `Node`, riproducendo una struttura comoda per poter presentare i contenuti di un nodo e semplificare lo spostamento fra frame associati e con relazioni di tipo gerarchico.
- **Attributi**:
  - `- content: String`
    Attributo che rappresenta i dati del contenuto del nodo.

  - `- height: Number`
    Attributo che rappresenta l'altezza del contenuto del nodo.
  - `- id: String`
    Attributo che rappresenta l'identificativo del contenuto.
  - `- type: String`
    Attributo che rappresenta il tipo di contenuto del nodo.
  - `- width: Number`
    Attributo che rappresenta la larghezza del contenuto del nodo.
  - `- x: Number`
    Attributo che rappresenta la posizione rispetto all'asse delle x del contenuto del nodo.
  - `- y: Number`
    Attributo che rappresenta la posizione rispetto all'asse delle y del contenuto del nodo.

- **Metodi**:
  - `+ getContent(): String`
    Metodo che restituisce il valore dell'attributo `content`.
  - `+ getHeight(): Number`
    Metodo che restituisce il valore dell'attributo `height`.
  - `+ getId(): String`
    Metodo che restituisce il valore dell'attributo `id`.
  - `+ getStyle(): Object`
    Metodo che restituisce un oggetto contenente le informazioni riguardanti lo stile CSS del contenuto del nodo. L'oggetto conterrà i seguenti campi dati: `top`, `left`, `width`, `height` e `position`.
  - `+ getType(): String`
    Metodo che restituisce il valore dell'attributo `type`.
  - `+ getWidth(): Number`
    Metodo che restituisce il valore dell'attributo `width`.
  - `+ getX(): Number`
    Metodo che restituisce il valore dell'attributo `x`.
  - `+ getY(): Number`
    Metodo che restituisce il valore dell'attributo `Y`.
  - `+ NodeContent(id: String, content: String, x: Number, y: Number, height: Number, width: Number, type: String)`
    Costruttore di `NodeContent`. Crea un'istanza con contenuto, posizione rispetto all'asse delle x e delle y, altezza, larghezza e tipo di contenuto pari ai valori dei parametri `id`, `content`, `x`, `y`, `height`, `width` e `class`.
    Parametri:
    * `id: String`
      Parametro che rappresenta l'`id` da dare all'oggetto.
    * `content: String`
      Parametro che rappresenta il contenuto dell'oggetto.
    * `x: Number`
      Parametro che rappresenta lo spostamento rispetto l'asse x dell'oggetto.
    * `y: Number`
      Parametro che rappresenta lo spostamento rispetto l'asse y dell'oggetto
    * `height: Number`
      Parametro che rappresenta l'altezza dell'oggetto.

    * `width: Number`
      Parametro che rappresenta la larghezza dell'oggetto.
    * `type: String`
      Parametro che rappresenta il tipo di contenuto presente nel campo `content` dell'oggetto.
- `+ setContent(content: String): void`
  Metodo che imposta il valore del campo dati `content`.
  Parametri:
    * `content: String`
      Parametro che rappresenta il valore da assegnare.
- `+ setHeight(h: Number): void`
  Metodo che imposta il valore del campo dati `height`.
  Parametri:
    * `h: Number`
      Parametro che rappresenta il valore da assegnare.
- `+ setWidth(w: Number): void`
  Metodo che imposta il valore del campo dati `width`.
  Parametri:
    * `w: Number`
      Parametro che rappresenta il valore da assegnare.
- `+ setX(x: Number): void`
  Metodo che imposta il valore del campo dati `x`.
  Parametri:
    * `x: Number`
      Parametro che rappresenta il valore da assegnare.
- `+ setY(y: Number): void`
  Metodo che imposta il valore del campo dati `y`.
  Parametri:
    * `y: Number`
      Parametro che rappresenta il valore da assegnare.

2.4.2.4 Premi::Front-End::Model::NodeReference

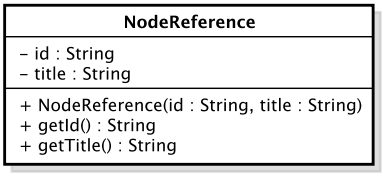


Figura 49: Premi::Front-End::Model::NodeReference

- **Descrizione**
  Rappresenta il riferimento ad un nodo della mappa mentale, contiene l'identificativo e il titolo del del nodo.
- **Utilizzo**
  Viene utilizzata per riferirsi ad un nodo della mappa mentale. Oggetti di questo tipo vengono usati per rappresentare sia i passi di un percorso di presentazione, sia per i nodi in relazione con un oggetto di tipo `PresentationNode`.

- **Relazioni con altre classi**:
  - *IN* `premiHierarchicalMenu`
    Rappresenta il menù che durante la presentazione permette di navigare la mappa mentale seguendo le relazioni gerarchiche presenti tra i vari nodi della mappa mentale. Questo componente consiste in una lista di elementi selezionabili dall'utente, ogni elemento corrisponde ad una relazione presente tra il nodo corrente ed un altro nodo della mappa.
  - *IN* `premiPath`
    Rappresenta il componente grafico che permette all'utente di modificare il percorso selezionato: togliendo dei nodi oppure rinominandolo. Questo componente visualizza la lista dei nodi presenti nel percorso. Per ogni nodo viene visualizzato il titolo del nodo ed un pulsante che permette di rimuoverlo dal percorso.
  - *IN* `premiSmartMenu`
    Rappresenta il menù che durante la presentazione permette di navigare la mappa mentale seguendo le associazioni definite tra i vari nodi. Questo componente consiste in una lista di elementi selezionabili dall'utente, ogni elemento corrisponde ad un'associazione presente tra il nodo corrente ed un altro nodo della mappa.
  - *IN* `Path`
    Rappresenta un percorso di presentazione costituito da una sequenza ordinata di passi.
  - *IN* `PresentationNode`
    Classe che estende `Node`, riproducendo una struttura comoda per poter presentare i contenuti di un nodo e semplificare lo spostamento fra frame associati e con relazioni di tipo gerarchico.
- **Attributi**:
  - `- id: String`
    Attributo che rappresenta l'identificativo del nodo associato al passo di percorso di presentazione.
  - `- title: String`
    Attributo che rappresenta il titolo del nodo nel passo del percorso di presentazione.
- **Metodi**:
  - `+ getId(): String`
    Metodo che restituisce l'identificativo del nodo associato al passo di percorso di presentazione.
  - `+ getTitle(): String`
    Metodo che restituisce il titolo del passo di percorso di presentazione.
  - `+ NodeReference(id: String, title: String)`
    Metodo costruttore della classe.
    Parametri:
    * `id: String`
      Parametro che rappresenta l'`id` del nodo che compare in un percorso di presentazione.
    * `title: String`
      Parametro che rappresenta il titolo del nodo che compare in un percorso di presentazione.

2.4.2.5 Premi::Front-End::Model::Path

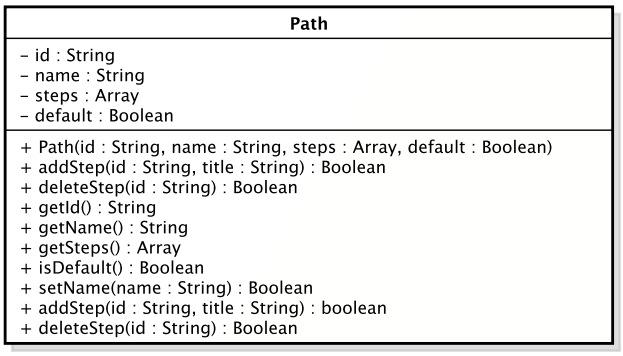


Figura 50: Premi::Front-End::Model::Path

- **Descrizione**
  Rappresenta un percorso di presentazione costituito da una sequenza ordinata di passi.
- **Utilizzo**
  Viene utilizzata per memorizzare un percorso di presentazione.
- **Relazioni con altre classi**:
  - *IN* `PathController`
    Classe che gestisce la visibilità e la logica della directive `premiPath`.
  - *IN* `PathsEditorController`
    Classe che gestisce le operazioni e la logica applicativa riguardante la gestione dei percorsi di presentazione definiti sulla mappa mentale.
  - *IN* `PathsListController`
    Classe che gestisce le operazioni e la logica applicativa della directive `premiPathsList`.
  - *IN* `PresentationViewerController`
    Classe che gestisce le operazioni e la logica applicativa riguardante l'esecuzione di un percorso di presentazione.
  - *IN* `premiPath`
    Rappresenta il componente grafico che permette all'utente di modificare il percorso selezionato: togliendo dei nodi oppure rinominandolo. Questo componente visualizza la lista dei nodi presenti nel percorso. Per ogni nodo viene visualizzato il titolo del nodo ed un pulsante che permette di rimuoverlo dal percorso.
  - *IN* `PathService`
    Questa classe si occupa del recupero e della modifica delle informazioni relative ai percorsi di presentazione relativi al progetto corrente.
  - *IN* `PathsEditorView`
    View dell'applicazione contenente la mappa mentale. Offre funzionalità legate ai percorsi di presentazione. Da questa view è possibile:
    * Creare/eliminare un percorso di presentazione;
    * Modificare un percorso aggiungendo e togliendo nodi, oppure rinominandolo;
    * Scegliere un percorso da presentare e successivamente presentarlo.
  - *IN* `PresentationView`
    View dell'applicazione che permette di effettuare la presentazione di un percorso scelto dall'utente. Utilizza la directive `premiPresentation` per offrire funzionalità legate alla presentazione lineare e due menù per permettere all'utente di spostarsi in modo non lineare, seguendo le relazioni definite tra i nodi della mappa mentale. Da questa vista è

inoltre possibile stampare ed esportare il percorso di presentazione in esecuzione, mediante le funzionalità offerte dal browser.

- *OUT* `NodeReference`
  Rappresenta il riferimento ad un nodo della mappa mentale, contiene l'identificativo e il titolo del del nodo.

- **Attributi**:
  - `- default: Boolean`
    Campo dati che specifica se il percorso di presentazione è quello di default.
  - `- id: String`
    Attributo che rappresenta l'identificativo del percorso di presentazione.
  - `- name: String`
    Attributo che rappresenta il nome del percorso di presentazione.
  - `- steps: Array`
    Attributo che rappresenta la successione dei passi del percorso di presentazione.

- **Metodi**:
  - `+ addStep(id: String, title: String): Boolean`
    Metodo che inserisce il passo di presentazione, formato dai valori dei parametri `id` e `title`, rispettivamente, in coda alla successione dei passi del percorso di presentazione.
    Parametri:
    * `id: String`
      Parametro che rappresenta l'identificativo del nodo da associare al nuovo passo di percorso di presentazione.
    * `title: String`
      Parametro che rappresenta il titolo del passo di percorso di presentazione da aggiungere.
  - `+ deleteStep(id: String): Boolean`
    Metodo che rimuove l'ultimo passo del percorso di presentazione con id pari al valore del parametro `id` e restituisce `true` se e solo se avviene tale rimozione.
    Parametri:
    * `id: String`
      Parametro che rappresenta l'identificativo del nodo associato al passo di percorso di presentazione da rimuovere.
  - `+ getId(): String`
    Metodo che ritorna l'identificativo del percorso.
  - `+ getName(): String`
    Metodo che restituisce il nome del percorso di presentazione.
  - `+ getSteps(): Array`
    Metodo che restituisce la successione dei passi del percorso di presentazione.
  - `+ isDefault(): Boolean`
    Metodo *getter* per il campo dati `default`.
  - `+ Path(id: String, name: String, steps: Array, isDefault: Boolean)`
    Metodo costruttore della classe.
    Parametri:
    * `id: String`
      Parametro contenente l'identificativo dell'oggetto da creare.
    * `name: String`
      Parametro contente il nome del percorso.

    * `steps: Array`
      Parametro contenente un `Array` di oggetti `PathStep`.
    * `isDefault: Boolean`
      Parametro che specifica se il percorso è di default o meno.
  - `+ setName(name: String): Boolean`
    Metodo che imposta il nome del percorso di presentazione con il valore del parametro `name`.
    Parametri:
    * `name: String`
      Parametro che rappresenta il nuovo nome da associare al percorso di presentazione.

2.4.2.6 Premi::Front-End::Model::PresentationNode

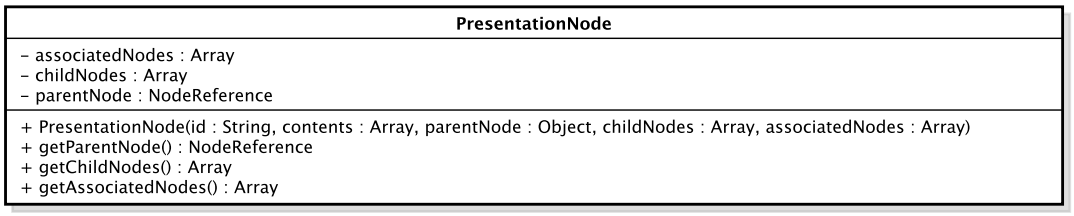


Figura 51: Premi::Front-End::Model::PresentationNode

- **Descrizione**
  Classe che estende `Node`, riproducendo una struttura comoda per poter presentare i contenuti di un nodo e semplificare lo spostamento fra frame associati e con relazioni di tipo gerarchico.
- **Utilizzo**
  Viene utilizzata per memorizzare i dati di un nodo e delle sue relazioni, informazioni sfruttate per creare una presentazione.
- **Classi ereditate**:
  - `Node`
- **Relazioni con altre classi**:
  - *IN* `PresentationViewerController`
    Classe che gestisce le operazioni e la logica applicativa riguardante l'esecuzione di un percorso di presentazione.
  - *IN* `premiNode`
    Rappresenta il componente grafico che visualizza il contenuto di un nodo. Questo contenuto non deve essere modificabile.
  - *IN* `PresentationService`
    Questa classe si occupa del recupero e della modifica delle informazioni relative alle presentazioni del progetto corrente.
  - *IN* `PresentationView`
    View dell'applicazione che permette di effettuare la presentazione di un percorso scelto dall'utente. Utilizza la directive `premiPresentation` per offrire funzionalità legate alla presentazione lineare e due menù per permettere all'utente di spostarsi in modo non lineare, seguendo le relazioni definite tra i nodi della mappa mentale. Da questa vista è inoltre possibile stampare ed esportare il percorso di presentazione in esecuzione, mediante le funzionalità offerte dal browser.

- *OUT* `NodeContent`
  Rappresenta il contenuto di un nodo. Contiene informazioni che indicano deve essere visualizzato il contenuto all'interno del nodo.
- *OUT* `NodeReference`
  Rappresenta il riferimento ad un nodo della mappa mentale, contiene l'identificativo e il titolo del del nodo.

- **Attributi**:
  - `- associatedNodes: Array`
    Campo dati che rappresenta un'array di oggetti di tipo `NodeReference` che si riferiscono ai nodi associati al nodo che si vuole creare.
  - `- childNodes: Array`
    Campo dati che rappresenta un'array di oggetti di tipo `NodeReference` che si riferiscono ai nodi figli del nodo che si vuole creare.
  - `- parentNode: NodeReference`
    Campo dati che rappresenta un riferimento al nodo di cui è figlio.

- **Metodi**:
  - `+ getAssociatedNodes(): Array`
    Metodo che restituisce un array contenente i vari `NodeReference` dei nodi associati a questo nodo.
  - `+ getChildNodes(): Array`
    Metodo che restituisce un array contenente i vari `NodeReference` dei nodi figli di questo nodo.
  - `+ getParentNode(): NodeReference`
    Metodo che restituisce un oggetto `NodeReference` relativo al nodo padre.
  - `+ PresentationNode(id: String, contents: Array, parentNode: Object, childNodes: Array, associatedNodes: Array)`
    Metodo costruttore della classe.
    Parametri:
    * `id: String`
      Parametro che rappresenta l'identificativo del nodo.
    * `contents: Array`
      Parametro che rappresenta i contenuti del nodo, l'array può contenere sia oggetti di tipo `NodeContent`, sia oggetti definiti con il JSON ricevuto in risposta dal back-end.
    * `parentNode: Object`
      Parametro che rappresenta un oggetto contenente le informazioni riguardanti il padre del nodo. I campi dati dell'oggetto sono definiti dal JSON ricevuto in risposta dal back-end.
    * `childNodes: Array`
      Parametro che rappresenta un'array di `Object`, definiti dal JSON ricevuto in risposta dal back-end, contenenti le informazioni riguardanti i figli del nodo che si vuole creare.
    * `associatedNodes: Array`
      Parametro che rappresenta un'array di `Object`, definiti dal JSON ricevuto in risposta dal back-end, contenenti le informazioni riguardanti i nodi associati al nodo che si vuole creare.

2.4.2.7 Premi::Front-End::Model::Project

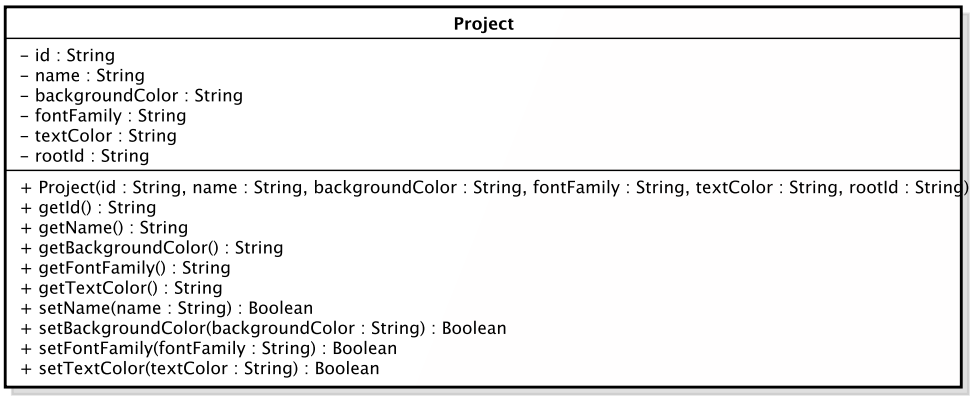


Figura 52: Premi::Front-End::Model::Project

- **Descrizione**
  Rappresenta un progetto creato dall'utente. Contiene le informazioni riguardanti i parametri globali del progetto: nome, formato del testo, colore di sfondo.
- **Utilizzo**
  Viene utilizzata per memorizzare i dati del progetto correntemente aperto nell'applicazione.
- **Relazioni con altre classi**:
  - *IN* `MindmapEditorController`
    Classe che gestisce le operazioni e la logica applicativa riguardante la modifica di una mappa.
  - *IN* `ProjectSettingsEditorController`
    Classe che si occupa di gestire la logica di funzionamento della directive `premiProjectSettingsEditor`.
  - *IN* `premiProjectSettingsEditor`
    Rappresenta il componente grafico che permette di modificare i dati del progetto aperto. Questo componente presenta i vari campi per interagire con i parametri del progetto e due pulsanti che l'utente utilizza per confermare o annullare le modifiche.
  - *IN* `premiProjectsList`
    Rappresenta il componente grafico che mostra la lista dei progetti creati dall'utente. Per ogni progetto della lista è presente un pulsante che permette di modificarlo, di cancellarlo e di presentarlo.
  - *IN* `ProjectService`
    Questa classe si occupa del recupero e della modifica delle informazioni riguardanti i progetti.
  - *IN* `DashboardView`
    View dell'applicazione contenente la lista dei progetti dell'utente correntemente autenticato. Da questa view è possibile:
    * Creare un nuovo progetto;
    * Aprire un progetto esistente per modificarlo oppure presentarlo;
    * Cancellare un progetto esistente.
  - *IN* `MindmapEditorView`
    View dell'applicazione contenente la mappa mentale. Offre funzionalità di modifica della mappa. Da questa view è possibile:
    * Aggiungere/togliere nodi;

    * Modificare il contenuto di un nodo;
    * Aggiungere/togliere associazioni tra nodi;
    * Modificare i parametri del progetto.

- **Attributi**:
  - `- backgroundColor: String`
    Attributo che rappresenta il colore di sfondo del progetto.
  - `- fontFamily: String`
    Attributo che rappresenta la famiglia del font del progetto.
  - `- id: String`
    Attributo che rappresenta l'identificativo del progetto.
  - `- name: String`
    Attributo che rappresenta il nome del progetto.
  - `- rootId: String`
    Campo dati contenente l'identificativo del nodo radice della mappa mentale presente all'interno del progetto.
  - `- textColor: String`
    Attributo che rappresenta il colore del testo del progetto.
- **Metodi**:
  - `+ getBackgroundColor(): String`
    Metodo che restituisce il nome del colore di sfondo associato al progetto, se quest'ultimo esiste, altrimenti restituisce `null`.
  - `+ getFontFamily(): String`
    Metodo che restituisce il nome della famiglia di font associato al progetto, se quest'ultimo esiste, altrimenti restituisce `null`.
  - `+ getId(): String`
    Metodo che restituisce l'identificativo associato al progetto, se quest'ultimo esiste, altrimenti restituisce `null`.
  - `+ getName(): String`
    Metodo che restituisce il nome associato al progetto, se quest'ultimo esiste, altrimenti restituisce `null`.
  - `+ getRootId(): String`
    Metodo che ritorna l'identificativo del nodo radice della mappa mentale associata al progetto.
  - `+ getTextColor(): String`
    Metodo che restituisce il nome del colore del testo associato al progetto, se quest'ultimo esiste, altrimenti restituisce `null`.
  - `+ Project(id: String, name: String, backgroundColor: String, fontFamily: String, textColor: String, rootId: String)`
    Metodo costruttore della classe.
    Parametri:
    * `id: String`
      Parametro che rappresenta l'identificativo del progetto.
    * `name: String`
      Parametro che rappresenta il nome del progetto.
    * `backgroundColor: String`
      Parametro che rappresenta il colore di sfondo dei frame presenti all'interno dei nodi del progetto.

    * `fontFamily: String`
      Parametro che rappresenta il font per il testo dei frame presenti all'interno dei nodi del progetto.
    * `textColor: String`
      Parametro che rappresenta il colore del testo dei frame presenti all'interno dei nodi del progetto.
    * `rootId: String`
      Parametro che rappresenta l'`id` del nodo radice della mappa mentale associata al progetto.

- `+ setBackgroundColor(backgroundColor: String): Boolean`
  Metodo che, se esiste un'istanza di progetto, sostituisce il colore di sfondo associato a tale istanza con il valore del parametro `backgroundColor` e restituisce `true`, altrimenti `false`.
  Parametri:
    * `backgroundColor: String`
      Parametro che rappresenta il colore di sfondo da sostituire a quello attuale del progetto esistente.
- `+ setFontFamily(fontFamily: String): Boolean`
  Metodo che, se esiste un'istanza di progetto, sostituisce la famiglia del font associato a tale istanza con il valore del parametro `fontFamily` e restituisce `true`, altrimenti `false`.
  Parametri:
    * `fontFamily: String`
      Parametro che rappresenta la famiglia di font da sostituire a quella attuale del progetto esistente.
- `+ setName(name: String): Boolean`
  Metodo che, se esiste un'istanza di progetto, sostituisce il nome associato a tale istanza con il valore del parametro `name` e restituisce `true`, altrimenti `false`.
  Parametri:
    * `name: String`
      Parametro che rappresenta il nome da sostituire a quello attuale del progetto esistente.
- `+ setTextColor(textColor: String): Boolean`
  Metodo che, se esiste un'istanza di progetto, sostituisce il colore del testo associato a tale istanza con il valore del parametro `textColor` e restituisce `true`, altrimenti `false`.
  Parametri:
    * `textColor: String`
      Parametro che rappresenta il colore del testo da sostituire a quello attuale del progetto esistente.

2.5 Premi::Front-End::Services

2.5.1 Informazioni generali

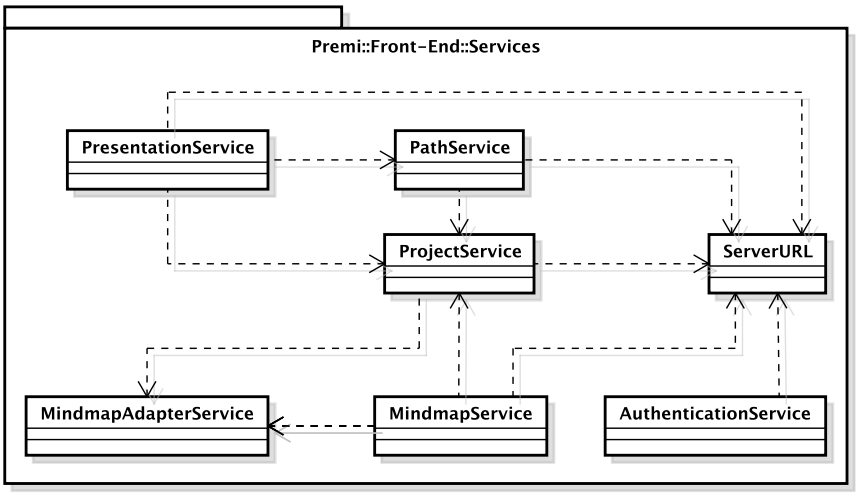


Figura 53: Premi::Front-End::Services

- **Descrizione**
  Package contenente le classi che descrivono il meccanismo con cui il front-end può interfacciarsi con le API REST del back-end. Permette di popolare il model del client e di richiedere l'esecuzione di operazioni al server.
- **Padre**: `Front-End`
- **Interazioni con altri componenti**:
  - `Controllers`
    Package contenente i controller della componente front-end dell'applicazione.
  - `Model`
    Package contenente le classi che definiscono la business logic dell'applicazione.

2.5.2 Classi

2.5.2.1 Premi::Front-End::Services::AuthenticationService

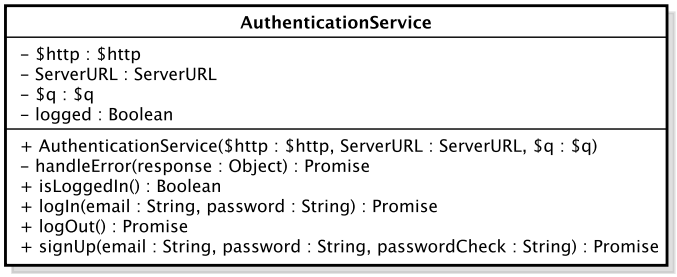


Figura 54: Premi::Front-End::Services::AuthenticationService

- **Descrizione**
  Questa classe si occupa di gestire il processo di autenticazione e di registrazione di un utente.

- **Utilizzo**
  Viene utilizzata per fornire ai controllers le funzionalità di autenticazione e registrazione di un utente. Se l'utente ha già effettuato il login fornisce informazioni riguardanti la sessione in corso.
- **Relazioni con altre classi**:
  - *IN* `AppRun`
    Classe che si occupa di gestire l'inizializzazione dell'applicazione.
  - *IN* `LoginController`
    Classe che gestisce le operazioni e la logica applicativa riguardante la pagina di login.
  - *IN* `RegistrationController`
    Classe che gestisce le operazioni e la logica applicativa riguardante la registrazione di un utente.
  - *OUT* `ErrorInfo`
    Rappresenta le informazioni di un errore che si è verificato eseguendo una determinata operazione.
  - *OUT* `ServerURL`
    Questa classe, realizzata come *Constant Recipe* di Angular, racchiude l'URL del server contenente il back-end dell'applicazione.
- **Attributi**:
  - `- logged: Boolean`
    Campo dati che specifica se l'utente corrente si è già autenticato.
  - `- ServerURL: ServerURL`
    Paramentro contenente un riferimento al servizio `SERVER_URL`.
  - `- $http: $http`
    Campo dati contenente un riferimento al servizio `$http` creato da *Angular*$_G$ per facilitare la comunicazione metiante protocollo HTTP.
  - `- $q: $q`
    Campo dati contenente un riferimento al servizio `$q` creato da *Angular*$_G$ per facilitare la gestione di funzione asincrone mediante l'utilizzo delle promesse.
- **Metodi**:
  - `+ AuthenticationService($http: $http, ServerURL: ServerURL, $q: $q)`
    Metodo costruttore della classe.
    Parametri:
    * `$http: $http`
      Parametro contenente un riferimento al servizio `$http` creato da *Angular*$_G$ per facilitare la comunicazione metiante protocollo HTTP.
    * `ServerURL: ServerURL`
      Campo dati contenente un riferimento al servizio `SERVER_URL`.
    * `$q: $q`
      Parametro contenente un riferimento al servizio `$q` creato da *Angular*$_G$ per facilitare la gestione di funzione asincrone mediante l'utilizzo delle promesse.
  - `- handleError(response: Object): Promise`
    Metodo di utilità che si occupa di gestire il fallimento di una chiamata HTTP verso il server.
    Parametri:
    * `response: Object`
      Parametro che rappresenta l'oggetto ricevuto in risposta da una richiesta HTTP effettuata utilizzando `$http`.

- `+ isLoggedIn(): Boolean`
  Metodo che ritorna il valore del campo dati `logged`.
- `+ logIn(email: String, password: String): Promise`
  Metodo che richiede al back-end l'autenticazione dell'utente, utilizzando i valori ricevuti come parametro. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga rifiutata, verrà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
  Parametri:
  * `email: String`
    Parametro che rappresenta l'email dell'utente che richiede l'accesso all'applicazione.
  * `password: String`
    Parametro che rappresenta la password dell'utente che richiede l'accesso all'applicazione.
- `+ logOut(): Promise`
  Metodo che richiede al back-end la de-autenticazione dell'utente. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga rifiutata, verrà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
- `+ signUp(email: String, password: String, passwordCheck: String): Promise`
  Metodo che, medinate `$http`, richiede al back-end la creazione di un nuovo account con le informazioni ricevute come parametro. Prima di effettuare la richiesta il metodo controlla che le due password ricevuto coincidano. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga rifiutata, verrà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
  Parametri:
  * `email: String`
    Parametro che rappresenta la mail dell'utente che si vuole registrare.
  * `password: String`
    Parametro che rappresenta la password dell'utente che si vuole registrare.
  * `passwordCheck: String`
    Parametro che rappresenta la conferma della password inserita dall'utente.

2.5.2.2 Premi::Front-End::Services::MindmapAdapterService

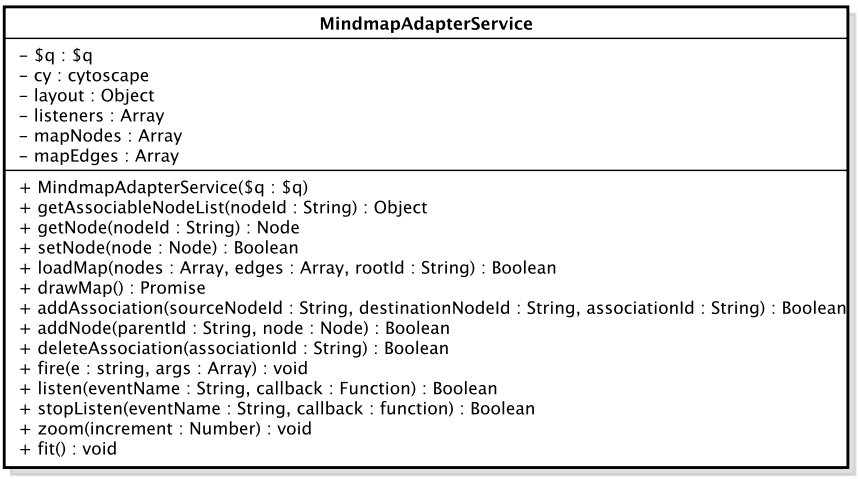


Figura 55: Premi::Front-End::Services::MindmapAdapterService

- **Descrizione**
  Questa classe si occupa di incapsulare le funzionalità offerte dalla libreria Cytoscape.js. Permette di rappresentare graficamente una mappa mentale e di accedere agevolmente ai dati in essa contenuti.
- **Utilizzo**
  Viene utilizzata per fornire ai services le funzionalità CRUD riguardanti i nodi di una mappa mentale e le relazioni tra di essi.
- **Relazioni con altre classi**:
  - *IN* `MindmapService`
    Questa classe si occupa del recupero e della modifica delle informazioni relative ai nodi e alle associazione presenti nella mappa mentale del progetto corrente. Utilizza `$http` per comunicare con il back-end e `MindmapAdapterService` per memorizzare i dati in locale.
  - *IN* `ProjectService`
    Questa classe si occupa del recupero e della modifica delle informazioni riguardanti i progetti.
- **Attributi**:
  - `- cy: cytoscape`
    Campo dati che rappresenta la mappa mentale associata al progetto corrente.
  - `- layout: Object`
    Campo dati contenente l'oggetto `layout` di Cytoscape.js.
  - `- listeners: Object`
    Campo dati contenente le funzioni registrate come gestori dei vari eventi disponibili. Le funzioni sono memorizzate all'interno dell'oggetto con delle coppie chiave/valore, usando come chiave il nome dell'evento e come valore un array di funzioni da invocare quando si verifica quel determinato evento.
  - `- mapEdges: Array`
    Campo dati contenente un'array con tutte le relazioni presenti all'interno dell'oggetto `cy`. L'array contiene degli oggetti definiti come `edge` di Cytoscape.js.

- - mapNodes: Array
  Campo dati contenente un'array con tutti i nodi presenti all'interno dell'oggetto cy. L'array contiene degli oggetti definiti come nodes di Cytoscape.js.
- - $q: $q
  Campo dati contenente un riferimento al servizio $q creato da *Angular*$_G$ per facilitare la gestione di funzione asincrone mediante l'utilizzo delle promesse.

- **Metodi**:
  - + addAssociation(sourceNodeId: String, destinationNodeId: String, associationId: String): Boolean
    Metodo che aggiunge un'associazione tra il nodo identificato da sourceNodeId e il nodo destinazione avente come id destinationNodeId. Se l'operazione va a buon fine restituisce true, altrimenti false.
    Parametri:
    * sourceNodeId: String
      Parametro che rappresenta l'id del nodo sorgente dell'associazione.
    * destinationNodeId: String
      Parametro che rappresenta l'id del nodo destinazione dell'associazione.
    * associationId: String
      Parametro che rappresenta l'id dell'associazione da inserire nella mappa mentale.
  - + addNode(parentId: String, node: Node): Boolean
    Metodo aggiungere il nodo node all'interno dell'oggetto _cy e successivamente crea una relazione gerarchica tra il nodo identificato da parentId e il nodo appena aggiunto. Sia il nodo, sia la relazione vengono poi aggiunti anche nelle due collezioni mapNodes e mapEdges. Ritorna un Booleano che specifica se l'operazione è andata a buon fine.
    Parametri:
    * parentId: String
      Parametro che rappresenta l'id del nodo padre della mappa mentale al quale aggiungere il nuovo nodo figlio.
    * node: Node
      Parametro che rappresenta il nodo da aggiungere alla mappa mentale.
  - + deleteAssociation(associationId: String): Boolean
    Metodo che rimuove l'associazione avente come id il valore del parametro associationId. Se l'operazione va a buon fine restituisce true, altrimenti false.
    Parametri:
    * associationId: String
      Parametro che rappresenta l'id dell'associazione da eliminare.
  - + drawMap(): Promise
    Metodo che si occupa di inizializzare l'oggetto _cy con i dati presenti nelle collezioni mapNodes e mapEdges. Una volta inizializzato l'oggetto _cy la mappa mentale diventerà visibile all'interno del presente nella view correntemente attiva[1]. Il metodo ritorna un oggetto Promise che può venire risolto o rifiutato. Nel caso la promessa venga rifiutata, verrà fornita come ragione un oggetto ErrorInfo contenente tutte le informazioni relative all'errore che si è verificato.
  - - fire(e: String, args: Array): void
    Metodo che notifica agli *observer* dell'evento e con i parametri presenti nell'oggetto args.
    Parametri:
    * e: String
      Parametro che rappresenta il nome dell'evento che si è verificato.

[1] Per maggiori informazioni riguardo il funzionamento di Cytoscape.js si rimanda all'appendice C

    * `args: Array`
      Parametro contenente le informazioni associate all'evento.

- `+ fit(): void`
  Metodo che adatta lo zoom della mappa mentale in modo che venga visualizzata completamente.

- `- getAssociableNodeList(nodeId: String): Object`
  Metodo che dato l'`id` di un nodo fornisce l'insieme dei nodi che possono essere associati ad esso. Tra due nodi può essere definita una sola associazione e, se tra un nodo e l'altro è già presente una relazione gerarchica, non è possibile aggiungere un'associazione. Le informazioni dei nodi sono memorizzate all'interno dell'oggetto ritornato sotto forma di coppie chiave/valore, usando come chiave l'`id` del nodo associabile e come valore il titolo del nodo.
  Parametri:
    * `nodeId: String`
      Parametro che rappresenta l'`id` del nodo per il quale si vuole recuperare la lista dei nodi associabili.

- `+ getNode(nodeId: String): Node`
  Metodo che crea un oggetto di tipo `Node` a partire i dati memorizzati dentro `_cy` con l'`id` uguale a `nodeId`. Se `nodeId` non è associato a nessun nodo di Cytoscape viene ritornato `null`.
  Parametri:
    * `nodeId: String`
      Parametro che rappresenta l'`id` del nodo del quale si vuole restituire il contenuto.

- `+ listen(eventName: String, callback: Function): Boolean`
  Metodo che permette di registrare una funzione come callback da invocare quando si verifica l'evento `eventName`. Per registrare le callback viene utilizzato l'omonimo metodo di `MindmapAdapterService`. Gli eventi disponibili e i relativi parametri con cui viene invocata la funzione di callback sono descritti nell'appendice C.
  Parametri:
    * `eventName: String`
      Parametro che rappresenta il nome dell'evento per il quale si vuole registrare la funzione di callback.
    * `callback: Function`
      Parametro che rappresenta la funzione di callback da associare all'evento.

- `+ loadMap(nodes: Array, edges: Array, rootId: String): Boolean`
  Metodo che legge i dati presenti nei due array `nodes` e `edges` e li trasforma nel formato di `cytoscape`. Una volta trasformati questi dati vengono memorizzati nei due array `mapNodes` e `mapEdges`.
  Parametri:
    * `nodes: Array`
      Parametro che rappresenta la collezione di nodi da caricare nella mappa mentale. Gli oggetti presenti nell'array sono definiti con il `JSON` ricevuto dalle API del back-end.
    * `edges: Array`
      Parametro che rappresenta la collezione le relazioni tra nodi da caricare nella mappa mentale. Gli oggetti presenti nell'array sono definiti con il `JSON` ricevuto dalle API del back-end.
    * `rootId: String`
      Parametro che rappresenta l'`id` del nodo radice della mappa mentale.

- `+ MindmapAdapterService($q: $q)`
  Metodo costruttore della classe.
  Parametri:
  * `$q: $q`
    Parametro contenente un riferimento al servizio `$q` creato da *Angular*$_G$ per facilitare la gestione di funzione asincrone mediante l'utilizzo delle promesse.
- `+ setNode(node: Node): Boolean`
  Metodo che legge dall'oggetto `node` l'`id` e va ad aggiornare i dati presenti nell'oggetto `_cy`, associati all'`id` del nodo, con quelli presenti in `node`. Se l'operazione va a buon fine restituisce `true`, altrimenti, se non è presente alcun nodo con l'`id` uguale a quello del nodo ricevuto per parametro restituisce `false`.
  Parametri:
  * `node: Node`
    Parametro contente l'oggetto da aggiornare all'interno di Cytoscape.
- `+ stopListen(eventName: String, callback: Function): Boolean`
  Metodo che permette di rimuovere una callback precendetemente registrata come gestore dell'evento `eventName`. Viene ritornato un valore `Boolean` che specifica se l'operazione è avvenuta con successo o meno.
  Parametri:
  * `eventName: String`
    Parametro che rappresenta il nome dell'evento che si vuole smettere di ascoltare.
  * `callback: Function`
    Parametro che rappresenta la callback da rimuovere.
- `+ zoom(increment: Number): void`
  Metodo che effettua lo zoom sulla mappa mentale.
  Parametri:
  * `increment: Number`
    Parametro che rappresenta di quanto incrementare (o decrementare, se il valore è negativo) lo zoom della mappa mentale.

2.5.2.3 Premi::Front-End::Services::MindmapService

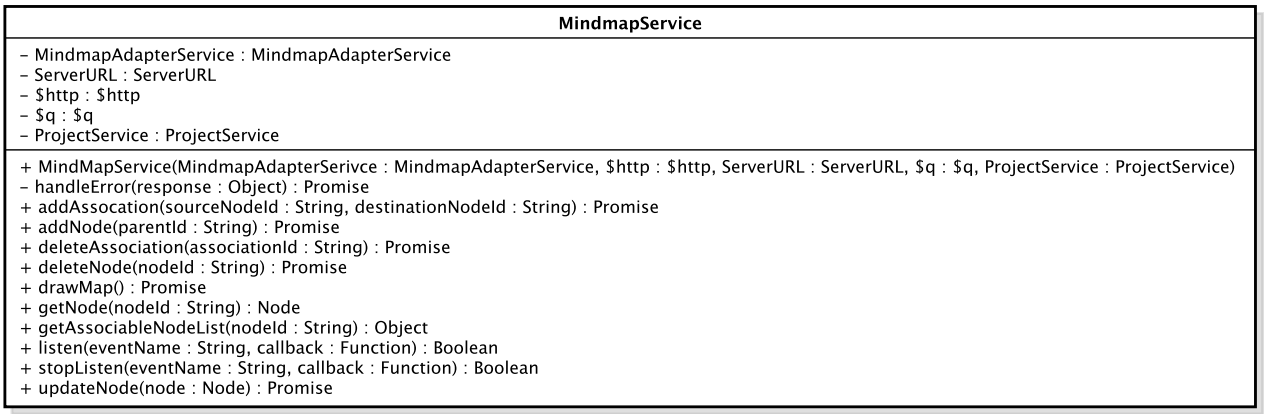


Figura 56: Premi::Front-End::Services::MindmapService

- **Descrizione**
  Questa classe si occupa del recupero e della modifica delle informazioni relative ai nodi e alle associazione presenti nella mappa mentale del progetto corrente. Utilizza `$http` per comunicare con il back-end e `MindmapAdapterService` per memorizzare i dati in locale.
- **Utilizzo**
  Viene utilizzata per fornire ai controllers la possibilità di interagire con i dati della mappa mentale, incapsulando tutte le informazioni riguardanti la logica organizzativa e di comunicazione con il back-end.
- **Relazioni con altre classi**:
  - *IN* `MindmapEditorController`
    Classe che gestisce le operazioni e la logica applicativa riguardante la modifica di una mappa.
  - *IN* `PathsEditorController`
    Classe che gestisce le operazioni e la logica applicativa riguardante la gestione dei percorsi di presentazione definiti sulla mappa mentale.
  - *OUT* `premiMindmap`
    Rappresenta il componente grafico che disegna la mappa mentale.
  - *OUT* `ErrorInfo`
    Rappresenta le informazioni di un errore che si è verificato eseguendo una determinata operazione.
  - *OUT* `Node`
    Rappresenta un nodo della mappa mentale. Contiene tutte le informazioni necessarie alla presentazione del contenuto del nodo.
  - *OUT* `MindmapAdapterService`
    Questa classe si occupa di incapsulare le funzionalità offerte dalla libreria Cytoscape.js. Permette di rappresentare graficamente una mappa mentale e di accedere agevolmente ai dati in essa contenuti.
  - *OUT* `ProjectService`
    Questa classe si occupa del recupero e della modifica delle informazioni riguardanti i progetti.
  - *OUT* `ServerURL`
    Questa classe, realizzata come *Constant Recipe* di Angular, racchiude l'URL del server contenente il back-end dell'applicazione.
- **Attributi**:
  - `- MindmapAdapterService: MindmapAdapterService`
    Campo dati contenente un riferimento al servizio `MindmapAdapterService`
  - `- ProjectService: ProjectService`
    Campo dati contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
  - `- ServerURL: ServerURL`
    Paramentro contenente un riferimento al servizio `SERVER_URL`.
  - `- $http: $http`
    Campo dati contenente un riferimento al servizio `$http` creato da *Angular*$_G$ per facilitare la comunicazione metiante protocollo HTTP.
  - `- $q: $q`
    Campo dati contenente un riferimento al servizio `$q` creato da *Angular*$_G$ per facilitare la gestione di funzione asincrone mediante l'utilizzo delle promesse.
- **Metodi**:

- `+ addAssociation(sourceNodeId: String, destinationNodeId: String): Promise`
  Metodo che aggiunge un'associazione tra un nodo sorgente avente come `id` il valore del parametro `sourceNodeId` e un nodo destinazione avente come `id` il valore del parametro `destinationNodeId`. La creazione dell'associazione viene effettuata sia in locale sia sul back-end, utilizzando rispettivamente `MindmapAdapterService` e `$http`. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga rifiutata, verrà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
  Parametri:
  * `sourceNodeId: String`
    Parametro che rappresenta l'`id` del nodo sorgente dell'associazione da creare.
  * `destinationNodeId: String`
    Parametro che rappresenta l'`id` del nodo destinazione dell'associazione da creare.
- `+ addNode(parentId: String): Promise`
  Metodo che aggiunge un nuovo nodo alla mappa mentale del progetto corrente. Il nuovo nodo verrà aggiunto come figlio di del nodo avente `id` pari al valore del parametro `parentId`. Il metodo come prima cosa richiede la creazione del nodo al back-end, dopodiché utilizza `MindmapAdapterService` per aggiungere il nodo in locale. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga risolta, verrà fornito come risultato un oggetto di tipo `Node` contenente tutte le informazioni del nuovo nodo. Se invece la promessa viene rifiutata, sarà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
  Parametri:
  * `parentId: String`
    Parametro che rappresenta l'id del nodo padre a cui aggiungere il nuovo nodo figlio.
- `+ deleteAssociation(associationId: String): Promise`
  Metodo che rimuove l'associazione avente come `id` il valore del parametro `associationId`. La cancellazione dell'associazione viene effettuata sia in locale sia sul back-end, utilizzando rispettivamente `MindmapAdapterService` e `$http`. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga rifiutata, verrà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
  Parametri:
  * `associationId: String`
    Parametro che rappresenta l'`id` dell'associazione da eliminare.
- `+ deleteNode(nodeId: String): Promise`
  Metodo che che richiede al back-end l'eliminazione del nodo avente l'`id` ricevuto come parametro dal progetto corrente. Una volta ricevuta la risposta `JSON` dal back-end vengono aggiornati i dati presenti di `MindmapService`. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga rifiutata, verrà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
  Parametri:
  * `nodeId: String`
    Parametro che rappresenta l'id del nodo da eliminare.
- `+ drawMap(): Promise`
  Metodo che i controllers invocano per disegnare la mappa mentale. Questo metodo si occupa di chiamare l'omonimo metodo offerto da `MindmapAdapterService`. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga rifiutata, verrà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.

- `+ getAssociableNodeList(nodeId: String): Object`
  Metodo che dato l'`id` di un nodo fornisce l'insieme dei nodi che possono essere associati ad esso. Tra due nodi può essere definita una sola associazione e, se tra un nodo e l'altro è già presente una relazione gerarchica, non è possibile aggiungere un'associazione. Le informazioni dei nodi sono memorizzate all'interno dell'oggetto ritornato sotto forma di coppie chiave/valore, usando come chiave l'`id` del nodo associabile e come valore il titolo del nodo.
  Parametri:
  * `nodeId: String`
    Parametro che rappresenta l'`id` del nodo per il quale si vuole recuperare la lista dei nodi associabili.
- `+ getNode(nodeId: String): Node`
  Metodo che restituisce il nodo avente `id` pari al valore del parametro `nodeId`. Se non è possibile creare il nodo viene restituito `null`.
  Utilizza `MindmapService` per richiedere la creazione dell'oggetto. La chiamata di questo metodo comporta l'aggiornamento del campo dati `node` della classe.
  Parametri:
  * `nodeId: String`
    Parametro che rappresenta l'`id` del nodo da recuperare.
- `- handleError(response: Object): Promise`
  Metodo di utilità che si occupa di gestire il fallimento di una chiamata HTTP verso il server.
  Parametri:
  * `response: Object`
    Parametro che rappresenta l'oggetto ricevuto in risposta ad una chiamata HTTP effettuata con `$http`.
- `+ listen(eventName: String, callback: Function): Boolean`
  Metodo permette di associare la funzione di callback ricevuta come parametro, all'evento `eventName`. Per registrare le callback viene utilizzato l'omonimo metodo di `MindmapAdapterService`. Gli eventi disponibili e i relativi parametri con cui viene invocata la funzione di callback sono descritti nell'appendice C.
  Parametri:
  * `eventName: String`
    Parametro che rappresenta il nome dell'evento per il quale si vuole registrare la funzione di callback.
  * `callback: Function`
    Parametro che rappresenta la funzione di callback da associare all'evento.
- `+ MindmapService($q: $q, $http: $http, MindmapAdapterService: MindmapAdapterService, ProjectService: ProjectService, ServerURL: ServerURL)`
  Metodo costruttore della classe.
  Parametri:
  * `$q: $q`
    Parametro contenente un riferimento al servizio `$q` creato da *Angular*$_G$ per facilitare la gestione di funzione asincrone mediante l'utilizzo delle promesse.
  * `$http: $http`
    Parametro contenente un riferimento al servizio `$http` creato da *Angular*$_G$ per facilitare la comunicazione metiante protocollo HTTP.
  * `MindmapAdapterService: MindmapAdapterService`
    Paramentro contenente un riferimento al servizio `MindmapAdapterService`

  * `ProjectService: ProjectService`
    Parametro contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
  * `ServerURL: ServerURL`
    Campo dati contenente un riferimento al servizio `SERVER_URL`.

- `+ stopListen(eventName: String, callback: Function): Boolean`
  Metodo che permette di rimuovere callback precendetemente registrata come gestore dell'evento `eventName`. Per rimuovere la funzione è necessario fornire la `key` con la quale è stata registrata. Viene ritornato un valore `Boolean` che specifica se l'operazione è avvenuta con successo o meno.
  Parametri:
  * `eventName: String`
    Parametro che rappresenta il nome dell'evento che si vuole smettere di ascoltare.
  * `callback: Function`
    Parametro che rappresenta la callback da rimuovere.
- `+ updateNode(node: Node): Promise`
  Metodo che rende permanenti le modifiche subite dall'oggetto `node` ricevuto come parametro utilizzando `MindmapAdapterService` e inviando i dati al back-end con `$http`. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga rifiutata, verrà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
  Parametri:
  * `node: Node`
    Parametro contenente l'oggetto nodo modificato e da salvare sul back-end.

2.5.2.4 Premi::Front-End::Services::PathService

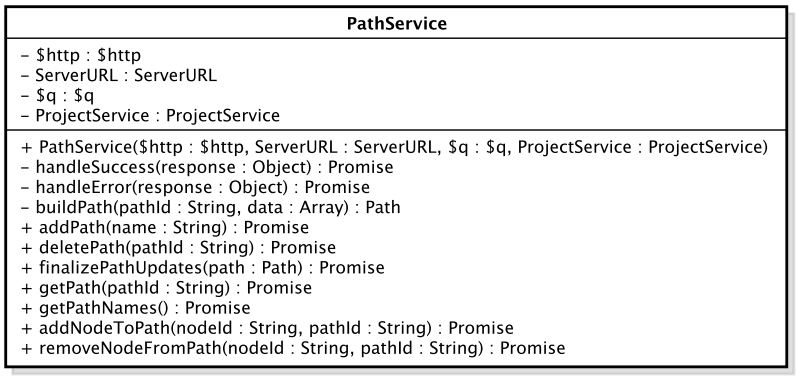


Figura 57: Premi::Front-End::Services::PathService

- **Descrizione**
  Questa classe si occupa del recupero e della modifica delle informazioni relative ai percorsi di presentazione relativi al progetto corrente.
- **Utilizzo**
  Viene utilizzata per fornire ai controllers le funzionalità CRUD riguardanti i percorsi di presentazione di un progetto.
- **Relazioni con altre classi**:

- *IN* `PathController`
  Classe che gestisce la visibilità e la logica della directive `premiPath`.
- *IN* `PathsEditorController`
  Classe che gestisce le operazioni e la logica applicativa riguardante la gestione dei percorsi di presentazione definiti sulla mappa mentale.
- *IN* `PathsListController`
  Classe che gestisce le operazioni e la logica applicativa della directive `premiPathsList`.
- *IN* `PresentationViewerController`
  Classe che gestisce le operazioni e la logica applicativa riguardante l'esecuzione di un percorso di presentazione.
- *IN* `PresentationService`
  Questa classe si occupa del recupero e della modifica delle informazioni relative alle presentazioni del progetto corrente.
- *OUT* `ErrorInfo`
  Rappresenta le informazioni di un errore che si è verificato eseguendo una determinata operazione.
- *OUT* `Path`
  Rappresenta un percorso di presentazione costituito da una sequenza ordinata di passi.
- *OUT* `ProjectService`
  Questa classe si occupa del recupero e della modifica delle informazioni riguardanti i progetti.
- *OUT* `ServerURL`
  Questa classe, realizzata come *Constant Recipe* di Angular, racchiude l'URL del server contenente il back-end dell'applicazione.

- **Attributi**:
  - `- ProjectService: ProjectService`
    Campo dati contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
  - `- ServerURL: ServerURL`
    Paramentro contenente un riferimento al servizio `SERVER_URL`.
  - `- $http: $http`
    Campo dati contenente un riferimento al servizio `$http` creato da *Angular*$_G$ per facilitare la comunicazione metiante protocollo HTTP.
  - `- $q: $q`
    Campo dati contenente un riferimento al servizio `$q` creato da *Angular*$_G$ per facilitare la gestione di funzione asincrone mediante l'utilizzo delle promesse.

- **Metodi**:
  - `+ addNodeToPath(nodeId: String, pathId: String): Promise`
    Metodo che richiede al back-end di aggiungere in coda il nodo identificato da `nodeId` al percorso `pathId`. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga risolta, verrà fornito come risultato un oggetto `Path` contenente le informazioni del percorso aggiornate. Se invece la promessa viene rifiutata, sarà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
    Parametri:
    * `nodeId: String`
      Parametro che rappresenta l'`id` del nodo da aggiungere.
    * `pathId: String`
      Parametro che rappresenta l'`id` del percorso a cui aggiungere il nodo.

- `+ addPath(name: String): Promise`
  Metodo che richiede al back-end la creazione di un nuovo percorso di presentazione di nome `name` all'interno del progetto corrente. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga risolta, verrà fornito come risultato un oggetto `Path` creato con i dati ricevuti in risposta dal back-end. Se invece la promessa viene rifiutata, sarà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
  Parametri:
  * `name: String`
    Parametro che rappresenta il nome del nuovo percorso di presentazione da creare.
- `- buildPath(pathId: String, data: Array): Path`
  Metodo di utilità che costruisce un oggetto di tipo `Path` a partire dai dati ricevuti come parametro.
  Parametri:
  * `pathId: String`
    Parametro che rappresenta l'identificativo del percorso da creare.
  * `data: Array`
    Parametro contenente un array di oggetti definiti con il JSON ottenuto in risposta dall'API `/projects/:projectId/paths/:pathId`.
- `+ deletePath(pathId: String): Promise`
  Metodo che richiede al server l'eliminazione del percorso di presentazione avente `id` pari al valore del parametro `pathId` dal progetto corrente. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga rifiutata, verrà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
  Parametri:
  * `pathId: String`
    Parametro che rappresenta il nome del percorso di presentazione da eliminare.
- `+ finalizePathUpdates(path: Path): Promise`
  Metodo che richiede al back-end il salvataggio delle modifiche subite dall'oggetto `path` ricevuto come parametro. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga rifiutata, verrà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
  Parametri:
  * `path: Path`
    Parametro che rappresenta l'oggetto modificato da salvare nel back-end.
- `+ getPath(pathId: String): Promise`
  Metodo che richiede al back-end i dati del percorso di presentazione identificato da `pathId`. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga risolta, verrà fornito come risultato un oggetto `Path` contenente le informazioni ricevute dal back-end. Se invece la promessa viene rifiutata, sarà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
  Parametri:
  * `pathId: String`
    Parametro che rappresenta `id` del percorso di presentazione da ottenere.
- `+ getPathNames(): Promise`
  Metodo che richiede al back-end la lista dei nomi dei percorsi di presentazione presenti nel progetto corrente. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga risolta, verrà fornito come risultato un array associativo contenente i nomi dei percorsi, associati al relativo id. Se invece la promessa viene rifiutata,

sarà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.

- `- handleError(response: Object): Promise`
  Metodo di utilità che si occupa di gestire il fallimento di una chiamata HTTP verso il server.
  Parametri:
  * `response: Object`
    Parametro che rappresenta l'oggetto ricevuto in risposta ad una chiamata HTTP effettuata con `$http`.
- `- handleSuccess(response: Object): Promise`
  Metodo di utilità che si occupa di gestire il successo di una chiamata HTTP verso il server e che non ritorna dati.
  Parametri:
  * `response: Object`
    Parametro che rappresenta l'oggetto ricevuto in risposta ad una chiamata HTTP effettuata con `$http`.
- `+ PathService($http: $http, $q: $q, ServerURL: ServerURL, ProjectService: ProjectService)`
  Metodo costruttore della classe.
  Parametri:
  * `$http: $http`
    Parametro contenente un riferimento al servizio `$http` creato da *Angular*$_G$ per facilitare la comunicazione metiante protocollo HTTP.
  * `$q: $q`
    Campo dati contenente un riferimento al servizio `$q` creato da *Angular*$_G$ per facilitare la gestione di funzione asincrone mediante l'utilizzo delle promesse.
  * `ServerURL: ServerURL`
    Campo dati contenente un riferimento al servizio `SERVER_URL`.
  * `ProjectService: ProjectService`
    Parametro contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
- `+ removeNodeFromPath(nodeId: String, pathId: String): Promise`
  Metodo che richiede al back-end di rimuovere il nodo identificato da `nodeId` dal percorso `pathId`. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga risolta, verrà fornito come risultato un oggetto `Path` contenente le informazioni del percorso aggiornate. Se invece la promessa viene rifiutata, sarà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
  Parametri:
  * `nodeId: String`
    Parametro che rappresenta l'`id` del nodo da rimuovere.
  * `pathId: String`
    Parametro che rappresenta l'`id` del percorso a cui aggiungere il nodo.

2.5.2.5 Premi::Front-End::Services::PresentationService

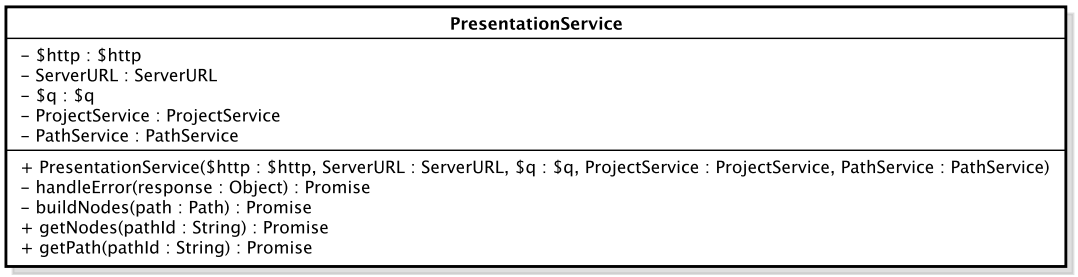


Figura 58: Premi::Front-End::Services::PresentationService

- **Descrizione**
  Questa classe si occupa del recupero e della modifica delle informazioni relative alle presentazioni del progetto corrente.
- **Utilizzo**
  Viene utilizzata per fornire ai controllers le funzionalità riguardanti la creazione di una presentazione basata su un percorso di presentazione definito all'interno di un progetto.
- **Relazioni con altre classi**:
  - *IN* `PresentationViewerController`
    Classe che gestisce le operazioni e la logica applicativa riguardante l'esecuzione di un percorso di presentazione.
  - *OUT* `ErrorInfo`
    Rappresenta le informazioni di un errore che si è verificato eseguendo una determinata operazione.
  - *OUT* `PresentationNode`
    Classe che estende `Node`, riproducendo una struttura comoda per poter presentare i contenuti di un nodo e semplificare lo spostamento fra frame associati e con relazioni di tipo gerarchico.
  - *OUT* `PathService`
    Questa classe si occupa del recupero e della modifica delle informazioni relative ai percorsi di presentazione relativi al progetto corrente.
  - *OUT* `ProjectService`
    Questa classe si occupa del recupero e della modifica delle informazioni riguardanti i progetti.
  - *OUT* `ServerURL`
    Questa classe, realizzata come *Constant Recipe* di Angular, racchiude l'URL del server contenente il back-end dell'applicazione.
- **Attributi**:
  - `- PathService: PathService`
    Campo dati contenente un riferimento al servizio che si occupa della gestione dei *percorsi*$_G$ di presentazione.
  - `- ProjectService: ProjectService`
    Campo dati contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
  - `- ServerURL: ServerURL`
    Paramentro contenente un riferimento al servizio `SERVER_URL`.
  - `- $http: $http`
    Campo dati contenente un riferimento al servizio `$http` creato da *Angular*$_G$ per facilitare la comunicazione metiante protocollo HTTP.

- `- $q: $q`
  Campo dati contenente un riferimento al servizio `$q` creato da *Angular*$_G$ per facilitare la gestione di funzione asincrone mediante l'utilizzo delle promesse.

- **Metodi**:
  - `- buildNodes(path: Path): Promise`
    Metodo d'utilità che a partire da un percorso di presentazione organizza i nodi della mappa mentale, in modo che sia possibile generare facilmente una presentazione.
    Parametri:
    * `path: Path`
      Parametro che rappresenta il percorso secondo il quale deve essere costruita la presentazione.
  - `+ getNodes(pathId: String): Promise`
    Metodo che richiede al back-end un array tutti i nodi presenti nella mappa mentale del progetto corrente e li ordina in modo che i primi nodi dell'array siano quelli presenti nel percorso `pathId`. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga risolta, verrà fornito come risultato un `Array` di `PresentationNode` da utilizzare per generare la presentazione. Se invece la promessa viene rifiutata, sarà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
    Parametri:
    * `pathId: String`
      Parametro che rappresenta l'`id` del percorso da presentare.
  - `+ getPath(pathId: String): Promise`
    Metodo che richiede al back-end il percorso di presentazione avente come `id` il valore presente nel parametro `pathId` appartenente al progetto corrente. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga risolta, verrà fornito come risultato un oggetto `Path` contenente i dati del percorso. Se invece la promessa viene rifiutata, sarà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
    Parametri:
    * `pathId: String`
      Parametro che rappresenta l'`id` del percorso di presentazione da ottenere.
  - `- handleError(response: Object): Promise`
    Metodo di utilità che si occupa di gestire il fallimento di una chiamata HTTP verso il server.
    Parametri:
    * `response: Object`
      Parametro che rappresenta l'oggetto ricevuto in risposta da una richiesta HTTP effettuata utilizzando `$http`.
  - `+ PresentationService($http: $http, $q: $q, ServerURL: ServerURL, ProjectService: ProjectService, PathService: PathService)`
    Metodo costruttore della classe.
    Parametri:
    * `$http: $http`
      Parametro contenente un riferimento al servizio `$http` creato da *Angular*$_G$ per facilitare la comunicazione metiante protocollo HTTP.
    * `$q: $q`
      Parametro contenente un riferimento al servizio `$q` creato da *Angular*$_G$ per facilitare la gestione di funzione asincrone mediante l'utilizzo delle promesse.

* `ServerURL: ServerURL`
  Campo dati contenente un riferimento al servizio `SERVER_URL`.
* `ProjectService: ProjectService`
  Parametro contenente un riferimento al servizio che si occupa della gestione dei *progetti*$_G$.
* `PathService: PathService`
  Parametro contenente un riferimento al servizio che si occupa della gestione dei *percorsi*$_G$ di presentazione.

2.5.2.6 Premi::Front-End::Services::ProjectService

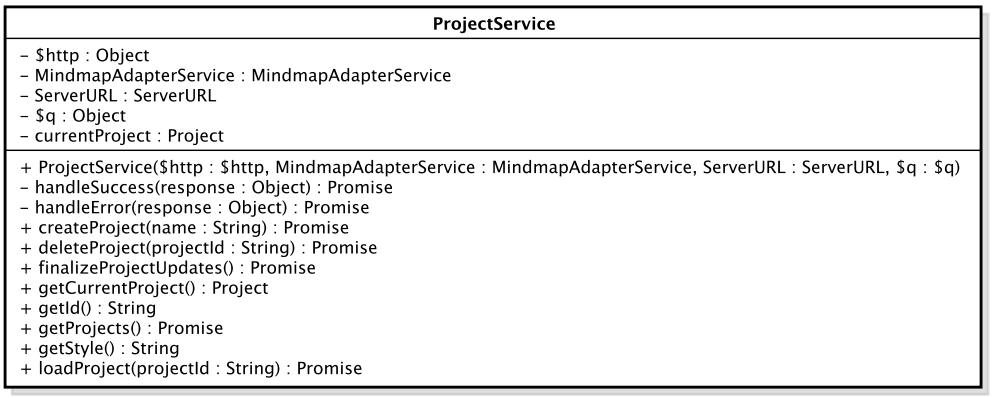


Figura 59: Premi::Front-End::Services::ProjectService

- **Descrizione**
  Questa classe si occupa del recupero e della modifica delle informazioni riguardanti i progetti.
- **Utilizzo**
  Viene utilizzata per fornire le funzionalità necessarie al recupero della lista dei progetti dell'utente correntemente autenticato. Permette di creare un nuovo progetto oppure di cancellare un progetto esistente.
- **Relazioni con altre classi**:
  - *IN* `AppRun`
    Classe che si occupa di gestire l'inizializzazione dell'applicazione.
  - *IN* `DashboardController`
    Classe che gestisce le operazioni e la logica applicativa riguardante la visualizzazione dei progetti di un utente.
  - *IN* `MindmapEditorController`
    Classe che gestisce le operazioni e la logica applicativa riguardante la modifica di una mappa.
  - *IN* `NodeContentsEditorController`
    Classe che si occupa di gestire la logica legata alla modifica del contenuto di un nodo.
  - *IN* `NodeController`
    Classe che si occupa di gestire il funzionamento della directive `premiNode`.
  - *IN* `ProjectSettingsEditorController`
    Classe che si occupa di gestire la logica di funzionamento della directive `premiProjectSettingsEditor`.

- *IN* `MindmapService`
  Questa classe si occupa del recupero e della modifica delle informazioni relative ai nodi e alle associazione presenti nella mappa mentale del progetto corrente. Utilizza `$http` per comunicare con il back-end e `MindmapAdapterService` per memorizzare i dati in locale.
- *IN* `PathService`
  Questa classe si occupa del recupero e della modifica delle informazioni relative ai percorsi di presentazione relativi al progetto corrente.
- *IN* `PresentationService`
  Questa classe si occupa del recupero e della modifica delle informazioni relative alle presentazioni del progetto corrente.
- *OUT* `ErrorInfo`
  Rappresenta le informazioni di un errore che si è verificato eseguendo una determinata operazione.
- *OUT* `Project`
  Rappresenta un progetto creato dall'utente. Contiene le informazioni riguardanti i parametri globali del progetto: nome, formato del testo, colore di sfondo.
- *OUT* `MindmapAdapterService`
  Questa classe si occupa di incapsulare le funzionalità offerte dalla libreria Cytoscape.js. Permette di rappresentare graficamente una mappa mentale e di accedere agevolmente ai dati in essa contenuti.
- *OUT* `ServerURL`
  Questa classe, realizzata come *Constant Recipe* di Angular, racchiude l'URL del server contenente il back-end dell'applicazione.

- **Attributi**:
  - `- currentProject: Project`
    Campo dati contenente un riferimento al progetto correntemente attivo.
  - `- MindmapAdapterService: MindmapAdapterService`
    Campo dati contenente un riferimento al servizio `MindmapAdapterService`
  - `- ServerURL: ServerURL`
    Paramentro contenente un riferimento al servizio `SERVER_URL`.
  - `- $http: Object`
    Campo dati contenente un riferimento al servizio `$http` creato da *Angular*$_G$ per facilitare la comunicazione metiante protocollo HTTP.
  - `- $q: Object`
    Campo dati contenente un riferimento al servizio `$q` creato da *Angular*$_G$ per facilitare la gestione di funzione asincrone mediante l'utilizzo delle promesse.

- **Metodi**:
  - `+ createProject(name: String): Promise`
    Metodo che richiede al back-end la creazione di un nuovo progetto usando il nome ricevuto per parametro. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga risolta, verrà fornito come risultato un oggetto `Project` contenente i dati del nuovo progetto. Se invece la promessa viene rifiutata, sarà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
    Parametri:
    * `name: String`
      Parametro che rappresenta il nome del nuovo progetto da creare.

- `+ deleteProject(projectId: String): Promise`
  Metodo che richiede al back-end l'eliminazione del progetto avente `id` pari al valore del parametro `projectId`. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga rifiutata, verrà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
  Parametri:
  * `projectId: String`
    Parametro che rappresenta il nome del progetto da eliminare.
- `+ finalizeProjectUpdates(): Promise`
  Metodo che richiede al back-end il salvataggio delle modifiche subite dai parametri dell'oggetto `currentProject`. Restituisce un valore `Boolean` che specifica se l'operazione è andata a buon fine. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga rifiutata, verrà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
- `+ getCurrentProject(): Project`
  Metodo che restituisce un riferimento all'oggetto presente in `currentProject`. Se `currentProject` non è istanziato, restituisce `null`.
- `+ getId(): String`
  Metodo che restituisce l'`id` dell'oggetto `currentProject`. Se questa non esiste restituisce una stringa vuota.
- `+ getProjects(): Promise`
  Metodo che richiede al back-end l'`id`, il nome e percorsi di presentazione (solo id e nomi) di tutti i progetti creati dall'utente. Il metodo ritorna un oggetto `Promise` che può venire risolto o rifiutato. Nel caso la promessa venga risolta, verrà fornito come risultato un `Array` di oggetti instanziati utilizzando il `JSON` ricevuto in risposta dal back-end. Se invece la promessa viene rifiutata, sarà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
- `+ getStyle(): String`
  Metodo che restituisce una stringa contente le classi CSS che definiscono stile generale di `currentProject`. Se `currentProject` non è stato istanziato restituisce una stringa vuota.
- `- handleError(response: Object): Promise`
  Metodo di utilità che si occupa di gestire il fallimento di una chiamata HTTP verso il server.
  Parametri:
  * `response: Object`
    Parametro che rappresenta l'oggetto ricevuto in risposta ad una chiamata HTTP effettuata con `$http`.
- `- handleSuccess(response: Object): Promise`
  Metodo di utilità che si occupa di gestire il successo di una chiamata HTTP verso il server e che non ritorna dati.
  Parametri:
  * `response: Object`
    Parametro che rappresenta l'oggetto ricevuto in risposta ad una chiamata HTTP effettuata con `$http`.
- `+ loadProject(projectId: String): Promise`
  Metodo che carica in memoria il progetto avente `id` pari al valore del parametro `projectId`. Viene effettuata una richiesta alle API del back-end e con i dati ricevuti viene prima creato un oggetto `Project` che sarà memorizzato nel campo dati `currentProject`, dopodiché, sempre con gli stessi dati, viene inizializzato `MindmapService`. Il metodo ritorna un oggetto `Promise` che può venire risolto o

rifiutato. Nel caso la promessa venga rifiutata, verrà fornita come ragione un oggetto `ErrorInfo` contenente tutte le informazioni relative all'errore che si è verificato.
Parametri:

  * `projectId: String`
    Parametro che rappresenta l'`id` del progetto da caricare in memoria.

- `+ ProjectService($http: Object, $q: Object, ServerURL: ServerURL, MindmapAdapterService: MindmapAdapterService)`
  Metodo costruttore della classe.
  Parametri:

  * `$http: Object`
    Parametro contenente un riferimento al servizio `$http` creato da *Angular*$_G$ per facilitare la comunicazione metiante protocollo HTTP.
  * `$q: Object`
    Parametro contenente un riferimento al servizio `$q` creato da *Angular*$_G$ per facilitare la gestione di funzione asincrone mediante l'utilizzo delle promesse.
  * `ServerURL: ServerURL`
    Campo dati contenente un riferimento al servizio `SERVER_URL`.
  * `MindmapAdapterService: MindmapAdapterService`
    Paramentro contenente un riferimento al servizio `MindmapAdapterService`

2.5.2.7 Premi::Front-End::Services::ServerURL

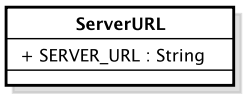


Figura 60: Premi::Front-End::Services::ServerURL

- **Descrizione**
  Questa classe, realizzata come *Constant Recipe* di Angular, racchiude l'URL del server contenente il back-end dell'applicazione.
- **Utilizzo**
  Viene utilizzata per fornire ai vari componenti dell'applicazione l'URL del server.
- **Relazioni con altre classi**:
  - *IN* `AuthenticationService`
    Questa classe si occupa di gestire il processo di autenticazione e di registrazione di un utente.
  - *IN* `MindmapService`
    Questa classe si occupa del recupero e della modifica delle informazioni relative ai nodi e alle associazione presenti nella mappa mentale del progetto corrente. Utilizza `$http` per comunicare con il back-end e `MindmapAdapterService` per memorizzare i dati in locale.
  - *IN* `PathService`
    Questa classe si occupa del recupero e della modifica delle informazioni relative ai percorsi di presentazione relativi al progetto corrente.
  - *IN* `PresentationService`
    Questa classe si occupa del recupero e della modifica delle informazioni relative alle presentazioni del progetto corrente.

  - *IN* `ProjectService`
    Questa classe si occupa del recupero e della modifica delle informazioni riguardanti i progetti.

- **Attributi**:
  - `+ SERVER_URL: String`
    Campo dati constante rappresentante l'URL del back-end.

2.6 Premi::Front-End::Views

2.6.1 Informazioni generali

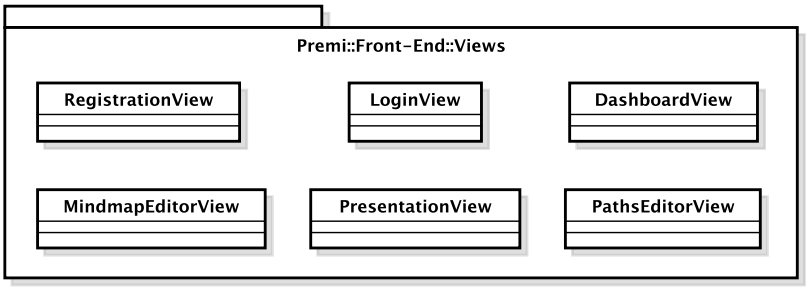


Figura 61: Premi::Front-End::Views

- **Descrizione**
  Package contenente le views della componente front-end dell'applicazione.
- **Padre**: `Front-End`
- **Interazioni con altri componenti**:
  - `Controllers`
    Package contenente i controller della componente front-end dell'applicazione.
  - `Directives`
    Package contenente le directives che compongno le views.
  - `Model`
    Package contenente le classi che definiscono la business logic dell'applicazione.

2.6.2 Classi

2.6.2.1 Premi::Front-End::Views::DashboardView

Figura 62: Premi::Front-End::Views::DashboardView

- **Descrizione**
  View dell'applicazione contenente la lista dei progetti dell'utente correntemente autenticato. Da questa view è possibile:

  - Creare un nuovo progetto;
  - Aprire un progetto esistente per modificarlo oppure presentarlo;
  - Cancellare un progetto esistente.

- **Utilizzo**
  Viene utilizzata come view iniziale dell'applicazione. L'utente, dopo aver effettuato l'accesso al proprio account, visualizza questa view per poter scegliere il progetto da modificare o presentare.
- **Relazioni con altre classi**:
  - *IN* `DashboardController`
    Classe che gestisce le operazioni e la logica applicativa riguardante la visualizzazione dei progetti di un utente.
  - *OUT* `premiProjectsList`
    Rappresenta il componente grafico che mostra la lista dei progetti creati dall'utente. Per ogni progetto della lista è presente un pulsante che permette di modificarlo, di cancellarlo e di presentarlo.
  - *OUT* `Project`
    Rappresenta un progetto creato dall'utente. Contiene le informazioni riguardanti i parametri globali del progetto: nome, formato del testo, colore di sfondo.
- **Attributi**:
  - `+ projects: Array`
    Campo dati contenente i vari progetti creati dall'utente.

2.6.2.2 Premi::Front-End::Views::LoginView

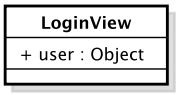


Figura 63: Premi::Front-End::Views::LoginView

- **Descrizione**
  View dell'applicazione contenente il form necessario per effettuare il login. Fornisce inoltre un link alla pagina di registrazione.
- **Utilizzo**
  Viene visualizzata all'avvio dell'applicazione per permettere all'utente di autenticarsi.
- **Relazioni con altre classi**:
  - *IN* `LoginController`
    Classe che gestisce le operazioni e la logica applicativa riguardante la pagina di login.
- **Attributi**:
  - `+ user: Object`
    Campo dati contenente un oggetto con due attributi `email` e `password`.

2.6.2.3 Premi::Front-End::Views::MindmapEditorView

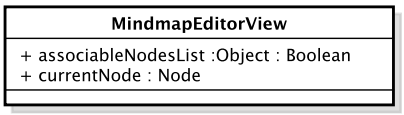


Figura 64: Premi::Front-End::Views::MindmapEditorView

- **Descrizione**
  View dell'applicazione contenente la mappa mentale. Offre funzionalità di modifica della mappa. Da questa view è possibile:
  - Aggiungere/togliere nodi;
  - Modificare il contenuto di un nodo;
  - Aggiungere/togliere associazioni tra nodi;
  - Modificare i parametri del progetto.
- **Utilizzo**
  Viene utilizzata per fornire all'utente la possibilità di interagire con la struttura della mappa e con le impostazioni del progetto.
- **Relazioni con altre classi**:
  - *IN* `MindmapEditorController`
    Classe che gestisce le operazioni e la logica applicativa riguardante la modifica di una mappa.
  - *OUT* `premiAssociationAdder`
    Rappresenta il componente grafico che permette all'utente di creare un'associazione tra il nodo selezionato e un altro nodo della mappa.
  - *OUT* `premiContextMenu`
    Rappresenta un menù contestuale generato in base agli oggetti passati nello scope isolato. Fornisce un pulsante per ogni oggetto ricevuto come parametro, ogni pulsante viene rappresentato con un'icona e con del testo. Al click di un pulsante viene invocata la funzione ad esso associata.
  - *OUT* `premiHeader`
    Rappresenta la barra di navigazione che permette all'utente di spostarsi tra le varie views dell'applicazione. Fornisce inoltre i pulsanti per visualizzare il manuale utente, effettuare il logout, chiudere al presentazione e il progetto. Inoltre, quando l'utente si trova nelle views in cui è possibile modificare il progetto, questa directive permette di mostrare la finestra di modifica delle impostazioni del progetto, sia di interagire con la visualizzazione della mappa mentale, modificando lo zoom e permettendo anche di stamparla.
  - *OUT* `premiNodeContentsEditor`
    Rappresenta il componente grafico che permette all'utente di modificare il contenuto di un nodo. Questo componente fornisce all'utente sia dei campi per inserire/modificare il contenuto del nodo, sia un'anteprima di come questo verrà visualizzato. Dall'anteprima sarà anche possibile spostare e ridimensionare i vari componenti. Saranno presenti inoltre due pulsanti, uno per confermare le modifiche e l'altro per annullarle e ripristinare lo stato del nodo.
  - *OUT* `Node`
    Rappresenta un nodo della mappa mentale. Contiene tutte le informazioni necessarie alla presentazione del contenuto del nodo.
  - *OUT* `Project`
    Rappresenta un progetto creato dall'utente. Contiene le informazioni riguardanti i parametri globali del progetto: nome, formato del testo, colore di sfondo.

- **Attributi**:
  - `+ associableNodesList: Object`
    Campo dati contenente le informazioni riguardanti i nodi della mappa mentale che possono essere associati al `currentNode`. Le informazioni all'interno dell'oggetto sono memorizzate come coppie chiave/valore, usando come chiave l'`id` del nodo e come valore il titolo.
  - `+ currentNode: Node`
    Campo dati sul quale effettuare le modifiche temporanee prima che l'utente le confermi. Viene usato anche per contenere i dati del nodo da visualizzare in anteprima.

2.6.2.4 Premi::Front-End::Views::PathsEditorView

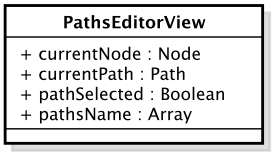


Figura 65: Premi::Front-End::Views::PathsEditorView

- **Descrizione**
  View dell'applicazione contenente la mappa mentale. Offre funzionalità legate ai percorsi di presentazione. Da questa view è possibile:
  - Creare/eliminare un percorso di presentazione;
  - Modificare un percorso aggiungendo e togliendo nodi, oppure rinominandolo;
  - Scegliere un percorso da presentare e successivamente presentarlo.
- **Utilizzo**
  Viene utilizzata per permettere all'utente di definire dei percorsi di presentazione e avviare la presentazione di uno di essi.
- **Relazioni con altre classi**:
  - *IN* `PathsEditorController`
    Classe che gestisce le operazioni e la logica applicativa riguardante la gestione dei percorsi di presentazione definiti sulla mappa mentale.
  - *OUT* `premiAddToPath`
    Rappresenta il componente grafico che permette all'utente di visualizzare il contenuto di un nodo e di aggiungerlo ad uno dei percorsi di presentazione esistenti. Questo componente visualizza una lista contenente tutti i percorsi disponibili al quale, selezionando quello desiderato, è possibile aggiungere il nodo slezionato.
  - *OUT* `premiHeader`
    Rappresenta la barra di navigazione che permette all'utente di spostarsi tra le varie views dell'applicazione. Fornisce inoltre i pulsanti per visualizzare il manuale utente, effettuare il logout, chiudere al presentazione e il progetto. Inoltre, quando l'utente si trova nelle views in cui è possibile modificare il progetto, questa directive permette di mostrare la finestra di modifica delle impostazioni del progetto, sia di interagire con la visualizzazione della mappa mentale, modificando lo zoom e permettendo anche di stamparla.
  - *OUT* `premiMindmap`
    Rappresenta il componente grafico che disegna la mappa mentale.

- *OUT* `premiPath`
  Rappresenta il componente grafico che permette all'utente di modificare il percorso selezionato: togliendo dei nodi oppure rinominandolo. Questo componente visualizza la lista dei nodi presenti nel percorso. Per ogni nodo viene visualizzato il titolo del nodo ed un pulsante che permette di rimuoverlo dal percorso.
- *OUT* `premiPathsList`
  Rappresenta il componente grafico che si occupa di visualizzare la lista dei percorsi di presentazione presenti nel progetto corrente. Dalla lista dei percorsi l'utente può selezionare un percorso per modificarlo, cancellare un determinato percorso oppure avviare la presentazione di quel percorso. Questo componente contiene anche un pulsante per creare un nuovo progetto.
- *OUT* `Node`
  Rappresenta un nodo della mappa mentale. Contiene tutte le informazioni necessarie alla presentazione del contenuto del nodo.
- *OUT* `Path`
  Rappresenta un percorso di presentazione costituito da una sequenza ordinata di passi.

- **Attributi**:
  - `+ currentNode: Node`
    Campo dati contenente le informazioni relative al nodo selezionato dall'utente.
  - `+ currentPath: Path`
    Campo dati contenente i dati del percorso selezionato dall'utente, viene utilizzato per memorizzare le modifiche effettuate dall'utente, ma che non sono ancora state confermate.
  - `+ pathSelected: Boolean`
    Campo dati che specifica se l'utente ha selezionato o meno un percorso di presentazione.
  - `+ pathsName: Array`
    Campo dati contenente le informazioni riguardanti i percorsi di presentazione creati dall'utente. Queste informazioni sono racchiuse in un oggetto con i seguenti attributi: `_id`, `name` e `default`.

2.6.2.5 Premi::Front-End::Views::PresentationView

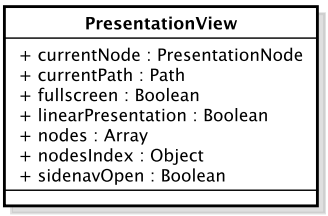


Figura 66: Premi::Front-End::Views::PresentationView

- **Descrizione**
  View dell'applicazione che permette di effettuare la presentazione di un percorso scelto dall'utente. Utilizza la directive `premiPresentation` per offrire funzionalità legate alla presentazione lineare e due menù per permettere all'utente di spostarsi in modo non lineare, seguendo le relazioni definite tra i nodi della mappa mentale. Da questa vista è inoltre possibile stampare ed esportare il percorso di presentazione in esecuzione, mediante le funzionalità offerte dal browser.

- **Utilizzo**
  Viene utilizzata per eseguire la presentazione del percorso scelto dall'utente e per permettergli di stamparlo, utilizzando l'apposita funzionalità del browser.
- **Relazioni con altre classi**:
  - *IN* `PresentationViewerController`
    Classe che gestisce le operazioni e la logica applicativa riguardante l'esecuzione di un percorso di presentazione.
  - *OUT* `premiHierarchicalMenu`
    Rappresenta il menù che durante la presentazione permette di navigare la mappa mentale seguendo le relazioni gerarchiche presenti tra i vari nodi della mappa mentale. Questo componente consiste in una lista di elementi selezionabili dall'utente, ogni elemento corrisponde ad una relazione presente tra il nodo corrente ed un altro nodo della mappa.
  - *OUT* `premiPresentation`
    Rappresenta il componente grafico che genera la presentazione.
  - *OUT* `premiSmartMenu`
    Rappresenta il menù che durante la presentazione permette di navigare la mappa mentale seguendo le associazioni definite tra i vari nodi. Questo componente consiste in una lista di elementi selezionabili dall'utente, ogni elemento corrisponde ad un'associazione presente tra il nodo corrente ed un altro nodo della mappa.
  - *OUT* `Path`
    Rappresenta un percorso di presentazione costituito da una sequenza ordinata di passi.
  - *OUT* `PresentationNode`
    Classe che estende `Node`, riproducendo una struttura comoda per poter presentare i contenuti di un nodo e semplificare lo spostamento fra frame associati e con relazioni di tipo gerarchico.
- **Attributi**:
  - `+ currentNode: PresentationNode`
    Campo dati contenente le informazioni relative al nodo correntemente visualizzato.
  - `+ currentPath: Path`
    Campo dati contenente le informazioni riguardo al percorso di visualizzazione che viene presentato.
  - `+ fullscreen: Boolean`
    Campo dati che specifica se la presentazione in corso è a schermo intero o meno.
  - `+ linearPresentation: Boolean`
    Campo dati che specifica se la presentazione in corso è lineare o meno.
  - `+ nodes: Array`
    Campo dati contenente una collezione di `PresentationNode` con tutti i nodi presenti nella mappa mentale, ordinati secondo il percorso di presentazione scelto dall'utente.
  - `+ nodesIndex: Object`
    Campo dati contenente le informazioni riguardante la posizione dei nodi nella sequenza della presentazione. Le informazioni all'interno dell'oggetto sono memorizzate come coppie chiave/valore, usando come chiave l'`id` del nodo e come valore un `Array` contenente tutte le posizioni dell'array `nodes` in cui compare quel determinato nodo.
  - `+ sidenavOpen: Boolean`
    Campo dati che specifica se il menù laterale è aperto o meno.

2.6.2.6 Premi::Front-End::Views::RegistrationView

Figura 67: Premi::Front-End::Views::RegistrationView

- **Descrizione**
  View dell'applicazione contenente il form per la registrazione di un nuovo utente. Contiene un link alla pagina di autenticazione.
- **Utilizzo**
  Viene visualizzata quando l'utente non possiede ancora un account e ha scelto di crearne uno.
- **Relazioni con altre classi**:
  - *IN* `RegistrationController`
    Classe che gestisce le operazioni e la logica applicativa riguardante la registrazione di un utente.
- **Attributi**:
  - `+ user: Object`
    Campo dati contenente un oggetto con i seguenti attributi: `email`, `password` e `passwordCheck`.

3 Specifica del Back-End

3.1 Premi::Back-End

3.1.1 Informazioni generali

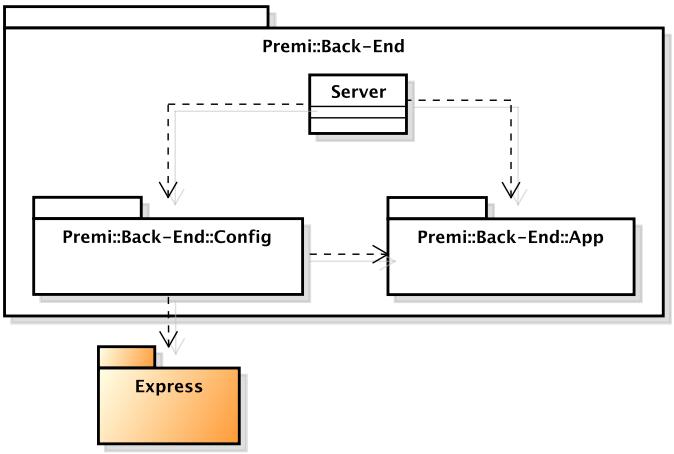


Figura 68: Premi::Back-End

- **Descrizione**
  Package contenente le componenti della parte back-end dell'applicazione.
- **Package contenuti**:
  - `App`
    Package contenente le componenti del server che implementano il pattern MVC.
  - `Config`
    Package contenente le componenti di configurazione del server.

3.1.2 Classi

3.1.2.1 Premi::Back-End::Server

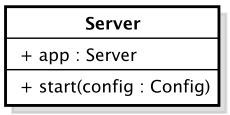


Figura 69: Premi::Back-End::Server

- **Descrizione**
  Classe che avvia il server. Nello specifico apre una connessione al database tramite Mongoose, invoca il middleware Express passando un riferimento al database MongoDB come parametro in modo che possa configurarsi con esso, invoca il middleware Passport ed infine si mette in ascolto su una determinata porta. È il componente client del design pattern Chain of responsibility. Utilizza i moduli Mongoose, Express, Passport e Chalk, quest'ultimo serve per definire lo stile dei tipi `String`.

- **Utilizzo**
  Utilizzato per avviare l'applicazione lato server. Inizializza, internamente al back-end, la catena di gestione delle chiamate REST utilizzando le classi contenute nel package `Routers`.
- **Relazioni con altre classi**:
  - *OUT* `ProjectRouter`
    Classe che gestisce le richieste relative alle operazioni riguardanti la mappa. Componente ConcreteHandler del design pattern Chain of responsibility.
  - *OUT* `StaticRouter`
    Classe che gestisce le richieste relative alle pagine statiche della componente back-end. Componente ConcreteHandler del design pattern Chain of responsibility.
  - *OUT* `UserRouter`
    Classe che gestisce le richieste relative alla registrazione e alla gestione della sessione di un utente. Componente ConcreteHandler del design pattern Chain of responsibility. Utilizza il modulo Passport.
  - *OUT* `Config`
    Questa classe gestisce la configurazione del server. *Non sono stati modellati attributi e metodi di questa classe in quanto viene gestita da Express.*
- **Attributi**:
  - `+ app: Server`
    Espone l'istanza di Express che rappresenta l'applicazione lato server.
- **Metodi**:
  - `+ start(config: Config)`
    Avvia il server sfruttando i parametri di configurazione contenuti nell'oggetto `config`.
    Parametri:
    * `config: Config`
      Rappresenta l'oggetto che contiene i dati di configurazione dell'applicazione.

3.2 Premi::Back-End::App

3.2.1 Informazioni generali

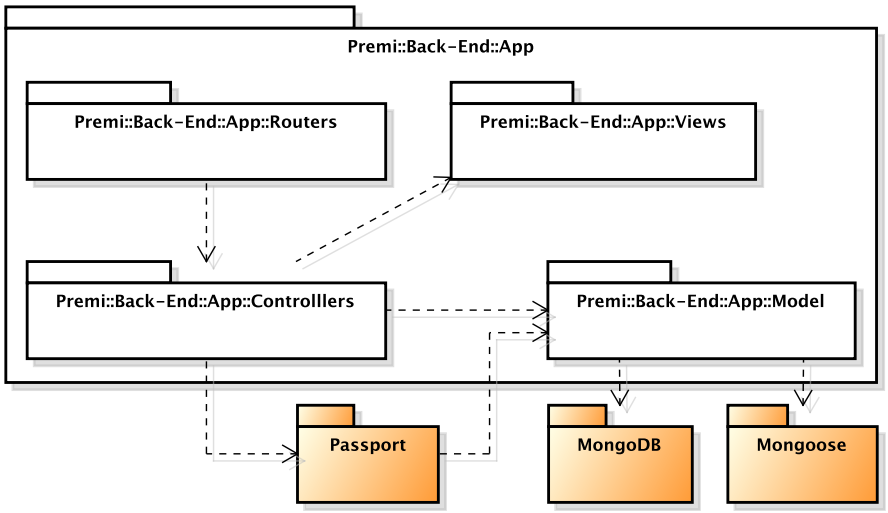


Figura 70: Premi::Back-End::App

- **Descrizione**
  Package contenente le componenti del server che implementano il pattern MVC.
- **Padre**: `Back-End`
- **Interazioni con altri componenti**:
  - `Config`
    Package contenente le componenti di configurazione del server.
- **Package contenuti**:
  - `Controllers`
    Package che contiene i controllers di Express, definisce la logica dell'applicazione.
  - `Models`
    Package contenente le classi che definiscono il model dell'applicazione. Queste classi sono definite come classi schema di Mongoose, il quale permette di utilizzare MongoDB tramite degli oggetti.
  - `Routers`
    Package contenente i router della componente back-end dell'applicazione. Contiene i file di configurazione relativi al routing delle richieste del client, ossia i routes di Express.
  - `Views`
    Package contenente le views della componente back-end dell'applicazione.

3.3 Premi::Back-End::App::Controllers

3.3.1 Informazioni generali

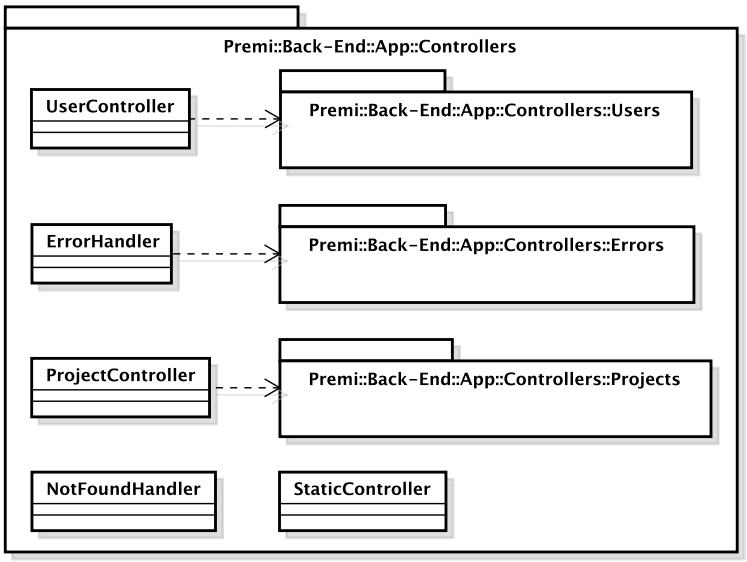


Figura 71: Premi::Back-End::App::Controllers

- **Descrizione**
  Package che contiene i controllers di Express, definisce la logica dell'applicazione.
- **Padre**: `App`
- **Interazioni con altri componenti**:
  - `Routers`
    Package contenente i router della componente back-end dell'applicazione. Contiene i file di configurazione relativi al routing delle richieste del client, ossia i routes di Express.
  - `Views`
    Package contenente le views della componente back-end dell'applicazione.
- **Package contenuti**:
  - `Errors`
    Package contenente i controllers per la gestione degli errori specifici.
  - `Projects`
    Package contenente i controllers relativi alla gestione della mappa mentale.
  - `Users`
    Package contenente i controllers relativi alla gestione dell'autenticazione dell'utente.

3.3.2 Classi

3.3.2.1 Premi::Back-End::App::Controllers::ErrorHandler

Figura 72: Premi::Back-End::App::Controllers::ErrorHandler

- **Descrizione**
  Classe middleware per la gestione degli errori. Ritorna al client un oggetto di tipo `Response` con stato HTTP 500 e descrizione dell'errore in formato JSON. È un componente ConcreteHandler del design pattern Chain of responsibility.
- **Utilizzo**
  Viene utilizzata quando si verifica un errore. Si preoccupa di delegare la costruzione del messaggio d'errore al modulo specifico qualora questo esista, altrimenti costruisce un messaggio d'errore generico. In questo modo i messaggi d'errore specifici vengono delegati ad un altro modulo, rendendo così possibile aggiungere in futuro altri moduli per gestire più flessibilmente nuove tipologie di errori.
- **Relazioni con altre classi**:
  - *IN* `ProjectRouter`
    Classe che gestisce le richieste relative alle operazioni riguardanti la mappa. Componente ConcreteHandler del design pattern Chain of responsibility.
  - *IN* `StaticRouter`
    Classe che gestisce le richieste relative alle pagine statiche della componente back-end. Componente ConcreteHandler del design pattern Chain of responsibility.
  - *IN* `UserRouter`
    Classe che gestisce le richieste relative alla registrazione e alla gestione della sessione di un utente. Componente ConcreteHandler del design pattern Chain of responsibility. Utilizza il modulo Passport.
  - *OUT* `PremiError`
    Classe di gestione degli errori. Esegue la costruzione del messaggio d'errore specifico per i moduli di `Premi::Back-End::App`.
- **Metodi**:
  - `+ handle(err: PremiError, req: Request, res: Response, next: function(PremiError))`
    Metodo che gestisce la costruzione dei messaggi d'errore ritornando un JSON contenente il messaggio d'errore.
    Parametri:
    * `err: PremiError`
      Rappresenta l'errore di tipo `PremiError`.
    * `req: Request`
      Rappresenta la richiesta inviata al server.
    * `res: Response`
      Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
    * `next: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

3.3.2.2 Premi::Back-End::App::Controllers::NotFoundHandler

Figura 73: Premi::Back-End::App::Controllers::NotFoundHandler

- **Descrizione**
  Classe che si occupa della gestione dell'errore di pagina non trovata. Componente ConcreteHandler del design pattern Chain of responsibility.
- **Utilizzo**
  Viene utilizzata per generare una pagina 404 di errore nel caso in cui l'URI passato non corrisponda ad una risorsa presente nell'applicazione.
- **Relazioni con altre classi**:
  - *IN* `ProjectRouter`
    Classe che gestisce le richieste relative alle operazioni riguardanti la mappa. Componente ConcreteHandler del design pattern Chain of responsibility.
  - *IN* `StaticRouter`
    Classe che gestisce le richieste relative alle pagine statiche della componente back-end. Componente ConcreteHandler del design pattern Chain of responsibility.
  - *IN* `UserRouter`
    Classe che gestisce le richieste relative alla registrazione e alla gestione della sessione di un utente. Componente ConcreteHandler del design pattern Chain of responsibility. Utilizza il modulo Passport.
- **Metodi**:
  - `+ handle(req: Request, res: Response)`
    Metodo che gestisce la costruzione dei messaggi d'errore ritornando un JSON contenente il messaggio d'errore.
    Parametri:
    * `req: Request`
      Rappresenta la richiesta inviata al server.
    * `res: Response`
      Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.

3.3.2.3 Premi::Back-End::App::Controllers::ProjectController

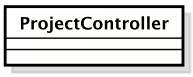


Figura 74: Premi::Back-End::App::Controllers::ProjectController

- **Descrizione**
  Classe che raggruppa i vari controllers responsabili delle operazioni riguardanti la mappa mentale attraverso `require`.
- **Utilizzo**
  Viene utilizzata per raggruppare i controllers responsabili della mappa mentale in modo da rendere più pulito il codice delle classi che utilizzano tali controllers. In questo modo le classi che vogliono

comunicare con i controllers della mappa mentale necessitano di includere solo questa classe e non ogni singolo controller.

- **Relazioni con altre classi**:
  - *IN* `ProjectRouter`
    Classe che gestisce le richieste relative alle operazioni riguardanti la mappa. Componente ConcreteHandler del design pattern Chain of responsibility.
  - *OUT* `NodeController`
    Classe che gestisce la logica applicativa riguardante la visualizzazione e la modifica dei nodi presenti in un progetto.
  - *OUT* `PathController`
    Classe che gestisce la logica applicativa riguardante la visualizzazione e la modifica dei percorsi di presentazione di un progetto.
  - *OUT* `ProjectManagementController`
    Classe che gestisce la logica applicativa riguardante la visualizzazione e la modifica dei progetti. Rappresenta il ConcreteHandler nel design pattern Chain of responsibility. Utilizza Passport.

3.3.2.4 Premi::Back-End::App::Controllers::StaticController

Figura 75: Premi::Back-End::App::Controllers::StaticController

- **Descrizione**
  Classe che gestisce le operazioni e la logica applicativa riguardante la visualizzazione di pagine HTML statiche. Rappresenta uno dei componenti ConcreteHandler del design pattern Chain of responsibility.
- **Utilizzo**
  Viene utilizzata per gestire la visualizzazione di pagine HTML statiche come il manuale utente.
- **Relazioni con altre classi**:
  - *IN* `StaticRouter`
    Classe che gestisce le richieste relative alle pagine statiche della componente back-end. Componente ConcreteHandler del design pattern Chain of responsibility.
  - *OUT* `UserManualView`
    Pagina HTML statica contenente il manuale dell'applicazione.
- **Metodi**:
  - `+ getUserManual(req: Request, res: Response)`
    Si occupa di restituire, attraverso il parametro `res`, la pagina del manuale utente richiesta dal client.
    Parametri:
    * `req: Request`
      Rappresenta la richiesta inviata dal client
    * `res: Response`
      Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.

3.3.2.5 Premi::Back-End::App::Controllers::UserController

Figura 76: Premi::Back-End::App::Controllers::UserController

- **Descrizione**
  Classe che raggruppa attraverso `require` i vari controllers responsabili delle operazioni legate alla gestione degli utenti. Si è scelto di predisporre questo raggruppamento per facilitare l'introduzione di nuove funzionalità legate alla gestione degli utenti.
- **Utilizzo**
  Viene utilizzata per raggruppare i controllers responsabili della gestione dei dati degli utenti. In questo modo le classi che vogliono comunicare con i controllers legati agli utenti necessitano di includere solo questa classe e non ogni singolo controller.
- **Relazioni con altre classi**:
  - *IN* `UserRouter`
    Classe che gestisce le richieste relative alla registrazione e alla gestione della sessione di un utente. Componente ConcreteHandler del design pattern Chain of responsibility. Utilizza il modulo Passport.
  - *OUT* `AuthenticationController`
    Classe che si occupa della registrazione e dell'autenticazione dell'utente nel sistema. È un componente ConcreteHandler del design pattern Chain of responsibility. Risulta essere il componente che eventualmente esegue la richiesta del client attraverso Passport.
  - *OUT* `AuthorizationController`
    Classe middleware che, utilizzando Passport, si occupa di controllare la consistenza dell'oggetto `session` durante la sessione associata all'utente autenticato. È un componente ConcreteHandler del design pattern Chain of responsibility.

3.4 Premi::Back-End::App::Controllers::Errors

3.4.1 Informazioni generali

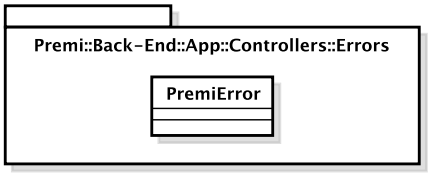


Figura 77: Premi::Back-End::App::Controllers::Errors

- **Descrizione**
  Package contenente i controllers per la gestione degli errori specifici.
- **Padre**: `Controllers`

3.4.2 Classi

3.4.2.1 Premi::Back-End::App::Controllers::Errors::PremiError

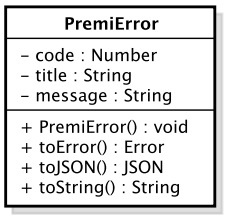


Figura 78: Premi::Back-End::App::Controllers::Errors::PremiError

- **Descrizione**
  Classe di gestione degli errori. Esegue la costruzione del messaggio d'errore specifico per i moduli di `Premi::Back-End::App`.
- **Utilizzo**
  Viene utilizzato da ErrorsHandler quando si verifica un errore specifico relativo alle classi di `Premi::Back-End::App`.
- **Relazioni con altre classi**:
  - *IN* `ErrorHandler`
    Classe middleware per la gestione degli errori. Ritorna al client un oggetto di tipo `Response` con stato HTTP 500 e descrizione dell'errore in formato JSON. È un componente ConcreteHandler del design pattern Chain of responsibility.
- **Attributi**:
  - `- code: Number`
    Campo dati contenente il codice dell'errore.
  - `- message: String`
    Campo dati contenente il messaggio che corrisponde all'errore.
  - `- title: String`
    Campo dati contenente il titolo dell'errore in forma di stringa
- **Metodi**:
  - `+ PremiError(err: Number)`
    Costruttore della classe `PremiError`.
    Parametri:
    * `err: Number`
      Rappresenta il codice identificativo dell'errore.
  - `+ toJSON(): JSON`
    Questo metodo ritorna l'errore in formato JSON.
  - `+ toString(): String`
    Effettua una concatenazione dei campi dati dell'errore e li ritorna in formato `String`.

3.5 Premi::Back-End::App::Controllers::Projects

3.5.1 Informazioni generali

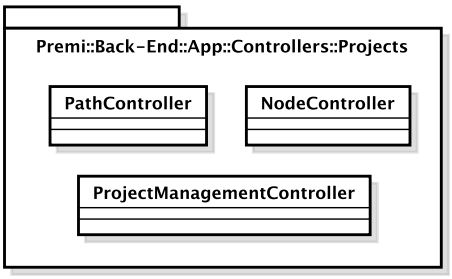


Figura 79: Premi::Back-End::App::Controllers::Projects

- **Descrizione**
  Package contenente i controllers relativi alla gestione della mappa mentale.
- **Padre**: `Controllers`

3.5.2 Classi

3.5.2.1 Premi::Back-End::App::Controllers::Projects::NodeController

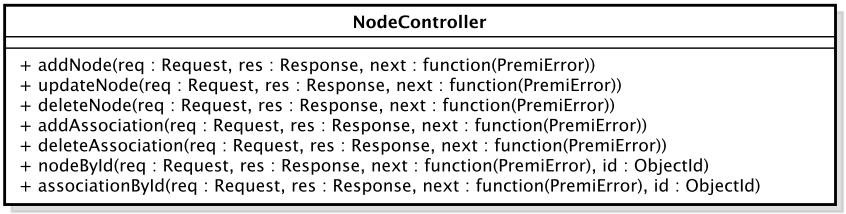


Figura 80: Premi::Back-End::App::Controllers::Projects::NodeController

- **Descrizione**
  Classe che gestisce la logica applicativa riguardante la visualizzazione e la modifica dei nodi presenti in un progetto.
- **Utilizzo**
  Viene utilizzata per implementare le funzionalità necessarie a gestire le richieste REST legate ai nodi presenti in un progetto.
- **Relazioni con altre classi**:
  - *IN* `ProjectController`
    Classe che raggruppa i vari controllers responsabili delle operazioni riguardanti la mappa mentale attraverso `require`.
  - *OUT* `ProjectModel`
    Questa classe rappresenta una mappa mentale, che contiene nodi, relazioni tra i nodi e percorsi di presentazione.
- **Metodi**:
  - `+ addAssociation(req: Request, res: Response, next: function(PremiError))`
    Aggiunge una relazione di tipo associazione tra due nodi presenti nella mappa mentale del progetto correntemente in uso dall'utente che ha effettuato il login.
    Parametri:

  * `req: Request`
    Rappresenta la richiesta inviata al server. Contiene l'identificativo dell'utente che ha effettuato la login. In `req` sono contenuti anche campi dati che rappresentano l'identificativo del progetto nel database, l'identificativo del nodo sorgente della relazione e quello del nodo destinazione.
  * `res: Response`
    Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
  * `next: function(PremiError)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

- `+ addNode(req: Request, res: Response, next: function(PremiError))`
  Aggiunge un nuovo nodo vuoto alla mappa mentale del progetto correntemente in uso dall'utente autenticato, specificandone il collegamento con il nodo padre.
  Parametri:
  * `req: Request`
    Rappresenta la richiesta inviata al server. Contiene l'identificativo dell'utente che ha effettuato la login, l'identificativo del progetto che sta modificando e l'identificativo del nodo di cui è figlio.
  * `res: Response`
    Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
  * `next: function(PremiError)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

- `+ associationById(req: Request, res: Response, next: function(PremiError), id: ObjectId)`
  Middleware sui parametri delle richieste REST relative all'id dell'associazione logica(non gerarchica). Controlla che l'id dell'associazione esista nel database: se ciò è verificato passa il controllo al successivo ConcreteHandler, altrimenti passa il controllo alla catena di gestione degli errori.
  Parametri:
  * `req: Request`
    Rappresenta la richiesta inviata al server.
  * `res: Response`
    Rappresenta la risposta che fornirà il server.
  * `next: function(PremiError)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.
  * `id: ObjectId`
    Rappresenta l'identificativo dell'associazione.

- `+ deleteAssociation(req: Request, res: Response, next: function(PremiError))`
  Rimuove una relazione di tipo `association` tra due nodi presenti nella mappa mentale del progetto correntemente in uso dall'utente che ha effettuato il login.
  Parametri:

  * `req: Request`
    È l'oggetto che rappresenta la richiesta inviata al server. Il metodo legge l'identificativo dell'utente che ha effettuato la login. In `req` sono contenuti anche campi dati che rappresentano l'identificativo del progetto nel database, l'identificativo del nodo sorgente della relazione e quello del nodo destinazione della relazione da eliminare.
  * `res: Response`
    È l'oggetto che rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
  * `next: function(PremiError)`
    Questo parametro rappresenta la callback che il metodo deve chiamare qualora si verificassero errori nell'esecuzione del metodo.

- `+ deleteNode(req: Request, res: Response, next: function(PremiError))`
  Rimuove un nodo dalla mappa mentale del progetto correntemente in uso dall'utente che ha effettuato il login. Restituisce la mappa mentale, comprendente nodi e relazioni come un oggetto JSON, priva del nodo rimosso e dell'eventuale sottoalbero di cui era padre.
  Parametri:
  * `req: Request`
    Rappresenta la richiesta inviata al server. Contiene l'identificativo dell'utente che ha effettuato la login, l'identificativo del progetto che sta modificando e l'identificativo del nodo da rimuovere dalla mappa mentale.
  * `res: Response`
    Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
  * `next: function(PremiError)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.
- `+ nodeById(req: Request, res: Response, next: function(PremiError), id: ObjectId)`
  Middleware sui parametri delle richieste REST relative all'id del nodo. Controlla che l'id del nodo esista nel database: se ciò è verificato passa il controllo al successivo ConcreteHandler, altrimenti passa il controllo alla catena di gestione degli errori.
  Parametri:
  * `req: Request`
    Rappresenta la richiesta inviata dal client.
  * `res: Response`
    Rappresenta la risposta che fornirà il server.
  * `next: function(PremiError)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.
  * `id: ObjectId`
    Rappresenta l'identificativo del nodo.
- `+ updateNode(req: Request, res: Response, next: function(PremiError))`
  Modifica il contenuto di un nodo della mappa mentale del progetto correntemente in uso dall'utente autenticato.
  Parametri:

* `req: Request`
  Rappresenta la richiesta inviata al server. Contiene l'identificativo dell'utente che ha effettuato la login. In `req` sono contenuti anche campi dati che rappresentano l'identificativo del progetto nel database, del nodo che il metodo deve modificare ed un oggetto JSON che raccoglie le modifiche da apportare al nodo.
* `res: Response`
  Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
* `next: function(PremiError)`
  Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

3.5.2.2 Premi::Back-End::App::Controllers::Projects::PathController

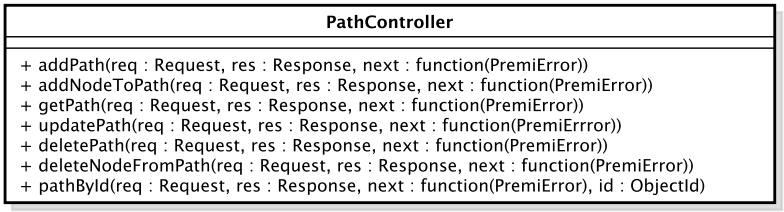


Figura 81: Premi::Back-End::App::Controllers::Projects::PathController

- **Descrizione**
  Classe che gestisce la logica applicativa riguardante la visualizzazione e la modifica dei percorsi di presentazione di un progetto.
- **Utilizzo**
  Viene utilizzata per implementare le funzionalità necessarie a gestire le richieste REST legate ai percorsi di presentazione.
- **Relazioni con altre classi**:
  - *IN* `ProjectController`
    Classe che raggruppa i vari controllers responsabili delle operazioni riguardanti la mappa mentale attraverso `require`.
  - *OUT* `ProjectModel`
    Questa classe rappresenta una mappa mentale, che contiene nodi, relazioni tra i nodi e percorsi di presentazione.
- **Metodi**:
  - `+ addNodeToPath(req: Request, res: Response, next: function(PremiError))`
    Aggiunge un nodo ad un percorso del progetto correntemente in uso dall'utente che ha effettuato il login.
    Parametri:
    * `req: Request`
      Rappresenta la richiesta inviata al server. Contiene l'identificativo dell'utente che ha effettuato la login, l'identificativo del progetto che sta modificando, l'identificativo del percorso da aggiornare e quello del nodo da inserire in esso.

  * `res: Response`
    Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
  * `next: function(PremiError)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

- `+ addPath(req: Request, res: Response, next: function(PremiError))`
  Crea un nuovo percorso di presentazione, con un nome specificato nel parametro `req`. Restituisce un oggetto JSON contenente l'identificativo, presente nel database, del percorso creato.
  Parametri:
  * `req: Request`
    Rappresenta la richiesta inviata al server. Contiene l'identificativo dell'utente che ha effettuato la login, l'identificativo del progetto che sta modificando e il nome del nuovo percorso da inserire nel database.
  * `res: Response`
    Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
  * `next: function(PremiError)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

- `+ deleteNodeFromPath(req: Request, res: Response, next: function(PremiError))`
  Rimuove un nodo da un percorso del progetto correntemente in uso dall'utente che ha effettuato il login.
  Parametri:
  * `req: Request`
    Rappresenta la richiesta inviata al server. Contiene l'identificativo dell'utente che ha effettuato la login, l'identificativo del progetto che sta modificando, l'identificativo del nodo da modificare e l'identificativo del nodo da rimuovere dal percorso.
  * `res: Response`
    Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
  * `next: function(PremiError)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

- `+ deletePath(req: Request, res: Response, next: function(PremiError))`
  Elimina un percorso di presentazione dal database.
  Parametri:
  * `req: Request`
    Rappresenta la richiesta inviata al server. Contiene l'identificativo dell'utente che ha effettuato la login, l'identificativo del progetto che sta modificando e l'identificativo del percorso da eliminare nel database.
  * `res: Response`
    Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.

    * `next: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

- `+ getPath(req: Request, res: Response, next: function(PremiError))`
  Restituisce un oggetto JSON contenente tutti i nodi facenti parte di un percorso di presentazione del progetto correntemente in uso dall'utente che ha effettuato il login.
  Parametri:
    * `req: Request`
      Rappresenta la richiesta inviata al server. Contiene l'identificativo dell'utente che ha effettuato la login, l'identificativo del progetto che sta modificando, l'identificativo del percorso di cui fornire l'insieme dei relativi nodi della mappa mentale.
    * `res: Response`
      Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
    * `next: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

- `+ pathById(req: Request, res: Response, next: function(PremiError), id: ObjectId)`
  Middleware sui parametri delle richieste REST relative all'id del path. Controlla che l'id del path esista nel database: se ciò è verificato passa il controllo al successivo ConcreteHandler, altrimenti passa il controllo alla catena di gestione degli errori.
  Parametri:
    * `req: Request`
      Rappresenta la richiesta inviata dal client.
    * `res: Response`
      Rappresenta la risposta che fornirà il server.
    * `next: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.
    * `id: ObjectId`
      Rappresenta l'identificativo del path.

- `+ updatePath(req: Request, res: Response, next: function(PremiError))`
  Questo metodo viene utilizzato per modificare le impostazioni di un percorso di presentazione nel database. In particolare è utilizzato per rinominare un percorso di presentazione. Ottiene il nuovo nome dal parametro `req`. Restituisce un messaggio di conferma o un errore, nel formato JSON stabilito.
  Parametri:
    * `req: Request`
      Rappresenta la richiesta inviata al server. Contiene legge l'identificativo dell'utente che ha effettuato la login, l'identificativo del progetto che sta modificando, l'identificativo del percorso di cui modificare il nome.
    * `res: Response`
      Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.

* `next: function(PremiError)`
  Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

3.5.2.3 Premi::Back-End::App::Controllers::Projects::ProjectManagementController

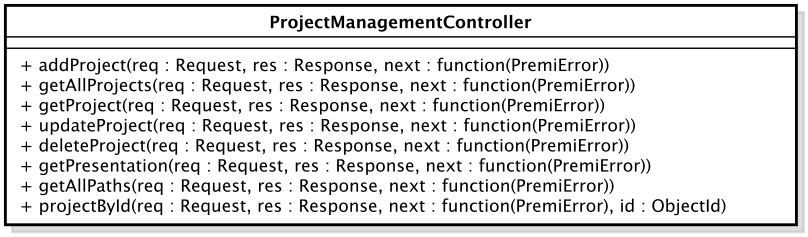


Figura 82: Premi::Back-End::App::Controllers::Projects::ProjectManagementController

- **Descrizione**
  Classe che gestisce la logica applicativa riguardante la visualizzazione e la modifica dei progetti. Rappresenta il ConcreteHandler nel design pattern Chain of responsibility. Utilizza Passport.
- **Utilizzo**
  Viene utilizzata per implementare le funzionalità necessarie a gestire le richieste REST legate ai progetti.
- **Relazioni con altre classi**:
  - *IN* `ProjectController`
    Classe che raggruppa i vari controllers responsabili delle operazioni riguardanti la mappa mentale attraverso `require`.
  - *OUT* `ProjectModel`
    Questa classe rappresenta una mappa mentale, che contiene nodi, relazioni tra i nodi e percorsi di presentazione.
- **Metodi**:
  - `+ addProject(req: Request, res: Response, next: function(PremiError))`
    Crea un nuovo progetto associato all'utente autenticato.
    Parametri:
    * `req: Request`
      Rappresenta la richiesta inviata al server.
    * `res: Response`
      Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
    * `next: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

- `+ deleteProject(req: Request, res: Response, next: function(PremiError))`
  Richiede al model la rimozione dal database di un progetto associato all'utente autenticato.
  Parametri:
  * `req: Request`
    Rappresenta la richiesta inviata al server. Contiene l'identificativo dell'utente autenticato. In `req` è contenuto anche un campo che rappresenta l'identificativo del progetto nel database che il metodo deve rimuovere.
  * `res: Response`
    Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
  * `next: function(PremiError)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.
- `+ getAllPaths(req: Request, res: Response, next: function(PremiError))`
  Restituisce i percorsi di presentazione del progetto relativo all'utente della sessione. Altrimenti restituisce un messaggio di errore.
  Parametri:
  * `req: Request`
    Rappresenta la richiesta inviata al server.
  * `res: Response`
    Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
  * `next: function(PremiError)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.
- `+ getAllProjects(req: Request, res: Response, next: function(PremiError))`
  Restituisce una lista contenente tutti i progetti relativi all'utente autenticato, a cui vengono associati i percorsi di presentazione creati, come un oggetto JSON.
  Parametri:
  * `req: Request`
    Rappresenta la richiesta inviata al server. Contiene l'identificativo dell'utente autenticato.
  * `res: Response`
    Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
  * `next: function(PremiError)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.
- `+ getPresentation(req: Request, res: Response, next: function(PremiError))`
  Questo metodo è utilizzato per fornire un oggetto JSON contenente i nodi della mappa mentale e le relazioni per il progetto specificato. Indica inoltre quali nodi sono compresi nel percorso di presentazione indicato.
  Parametri:

  * `req: Request`
    Rappresenta la richiesta inviata al server. Contiene l'identificativo dell'utente che ha effettuato la login e gli identificativi del progetto e del percorso di cui si vogliono ottenere i dati per la presentazione.
  * `res: Response`
    Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
  * `next: function(PremiError)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

- `+ getProject(req: Request, res: Response, next: function(PremiError))`
  Restituisce la mappa mentale comprendente nodi e relazioni in formato JSON. La mappa mentale restituita è associata ad un progetto dell'utente autenticato.
  Parametri:
  * `req: Request`
    Rappresenta la richiesta inviata al server. Contiene l'identificativo dell'utente autenticato. In `req` è contenuto anche un campo che rappresenta l'identificativo del progetto di cui si vuole ottenere la mappa mentale.
  * `res: Response`
    Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
  * `next: function(PremiError)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

- `+ projectById(req: Request, res: Response, next: function(PremiError), id: ObjectId)`
  Middleware sui parametri delle richieste REST relative all'id del progetto. Controlla che l'id del progetto esista nel database: se ciò è verificato inserisce l'id in `req` e passa il controllo al successivo ConcreteHandler, altrimenti passa il controllo alla catena di gestione degli errori.
  Parametri:
  * `req: Request`
    Rappresenta la richiesta inviata dal client.
  * `res: Response`
    Rappresenta la risposta che fornirà il server.
  * `next: function(PremiError)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.
  * `id: ObjectId`
    Rappresenta l'identificativo del progetto.

- `+ updateProject(req: Request, res: Response, next: function(PremiError))`
  Modifica le impostazioni del progetto associato all'utente autenticato. Modifica il nome del progetto, il colore dello sfondo, il colore del testo e la famiglia di font.
  Parametri:

* `req: Request`
  Rappresenta la richiesta inviata al server. Contiene l'identificativo dell'utente che ha effettuato la login e l'identificativo del progetto che si vuole rinominare e il nuovo nome da assegnargli.
* `res: Response`
  Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
* `next: function(PremiError)`
  Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

3.6 Premi::Back-End::App::Controllers::Users

3.6.1 Informazioni generali

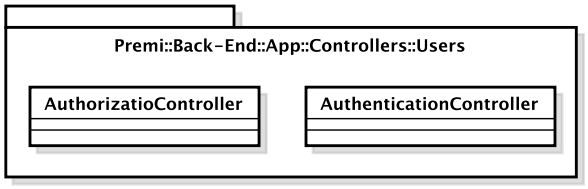


Figura 83: Premi::Back-End::App::Controllers::Users

- **Descrizione**
  Package contenente i controllers relativi alla gestione dell'autenticazione dell'utente.
- **Padre**: `Controllers`

3.6.2 Classi

3.6.2.1 Premi::Back-End::App::Controllers::Users::AuthenticationController

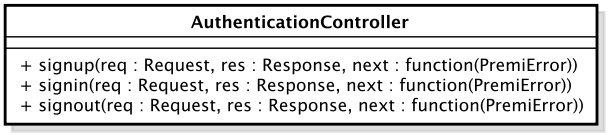


Figura 84: Premi::Back-End::App::Controllers::Users::AuthenticationController

- **Descrizione**
  Classe che si occupa della registrazione e dell'autenticazione dell'utente nel sistema. È un componente ConcreteHandler del design pattern Chain of responsibility. Risulta essere il componente che eventualmente esegue la richiesta del client attraverso Passport.
- **Utilizzo**
  Viene utilizzata per implementare le funzionalità necessarie a gestire le richieste REST legate alla registrazione e all'autenticazione dell'utente.
- **Relazioni con altre classi**:

- *IN* `UserController`
  Classe che raggruppa attraverso `require` i vari controllers responsabili delle operazioni legate alla gestione degli utenti. Si è scelto di predisporre questo raggruppamento per facilitare l'introduzione di nuove funzionalità legate alla gestione degli utenti.
- *OUT* `Session`
  Classe che gestisce la sessione utente dell'applicazione. *Non sono stati modellati attributi e metodi di questa classe in quanto viene inizializzata da Express ed utilizzata da Passport attraverso funzionalità interne ai due middleware.*
- *OUT* `UserModel`
  Classe che modella la creazione e la gestione dei dati utente.

- **Metodi**:
  - `+ signin(req: Request, res: Response, next: function(PremiError))`
    Esegue l'autenticazione attraverso Passport, aggiorna i dati della sessione e risponde al client con i dati non sensibili dell'utente che ha effettuato l'autenticazione.
    Parametri:
    * `req: Request`
      Rappresenta la richiesta inviata dal client, contiene la richiesta di login dell'utente.
    * `res: Response`
      Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
    * `next: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.
  - `+ signout(req: Request, res: Response)`
    Esegue il logout dell'utente dal sistema.
    Parametri:
    * `req: Request`
      Rappresenta la richiesta inviata dal client, contiene la richiesta di logout inviata dal client.
    * `res: Response`
      Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo.
  - `+ signup(req: Request, res: Response, next: function(PremiError))`
    Effettua la registrazione dell'utente nel sistema creando ed inserendo un nuovo document all'interno della collection `User`.
    Parametri:
    * `req: Request`
      Rappresenta la richiesta inviata dal client, contiene i dati che vengono inseriti nel nuovo document della collection `User` se tutti i campi dati del nuovo oggetto document creato sono corretti, altrimenti viene attivata la catena di gestione dell'errore.
    * `res: Response`
      Rappresenta la risposta che il server fornirà al termine dell'esecuzione del metodo,contiene le informazioni non sensibili che l'utente ha inserito nel database e con cui viene identificato oppure contiene un messaggio d'errore.
    * `next: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

3.6.2.2 Premi::Back-End::App::Controllers::Users::AuthorizationController

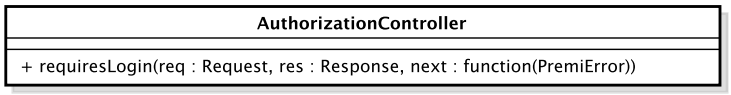


Figura 85: Premi::Back-End::App::Controllers::Users::AuthorizationController

- **Descrizione**
  Classe middleware che, utilizzando Passport, si occupa di controllare la consistenza dell'oggetto `session` durante la sessione associata all'utente autenticato. È un componente ConcreteHandler del design pattern Chain of responsibility.
- **Utilizzo**
  Componente middleware della catena, esegue controlli sullo stato della sessione relativa all'utente. Se l'utente non possiede i permessi adeguati o la sessione è scaduta restituisce un messaggio d'errore, altrimenti passa il controllo al prossimo ConcreteHandler che gestirà il normale flusso d'esecuzione del programma.
- **Relazioni con altre classi**:
  - *IN* `UserController`
    Classe che raggruppa attraverso `require` i vari controllers responsabili delle operazioni legate alla gestione degli utenti. Si è scelto di predisporre questo raggruppamento per facilitare l'introduzione di nuove funzionalità legate alla gestione degli utenti.
  - *OUT* `Session`
    Classe che gestisce la sessione utente dell'applicazione. *Non sono stati modellati attributi e metodi di questa classe in quanto viene inizializzata da Express ed utilizzata da Passport attraverso funzionalità interne ai due middleware.*
  - *OUT* `UserModel`
    Classe che modella la creazione e la gestione dei dati utente.
- **Metodi**:
  - `+ requiresLogin(req: Request, res: Response, next: function(PremiError))`
    Metodo usato dal middleware per verificare che l'utente che esegue una richiesta sia effettivamente un utente autenticato.
    Parametri:
    * `req: Request`
      Rappresenta la richiesta inviata dal client.
    * `res: Response`
      Rappresenta la risposta che fornirà il server.
    * `next: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione per passare il controllo ai successivi middleware. La presenza del parametro facoltativo `PremiError` attiva la catena di gestione dell'errore in sostituzione della normale catena di gestione delle richieste.

3.7 Premi::Back-End::App::Models

3.7.1 Informazioni generali

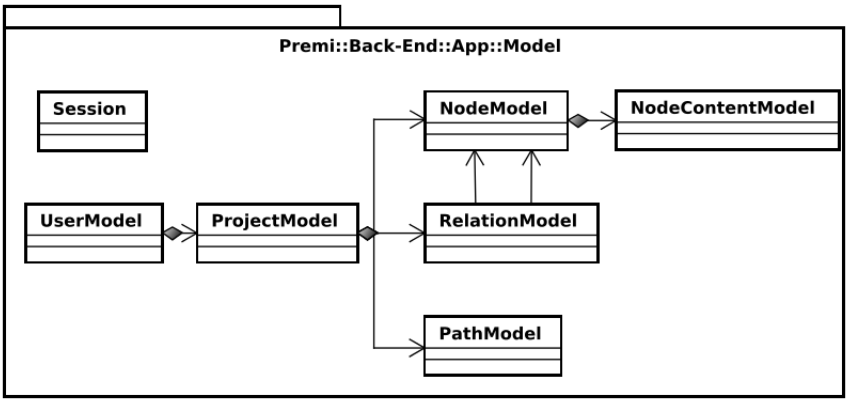


Figura 86: Premi::Back-End::App::Models

- **Descrizione**
  Package contenente le classi che definiscono il model dell'applicazione. Queste classi sono definite come classi schema di Mongoose, il quale permette di utilizzare MongoDB tramite degli oggetti.
- **Padre**: `App`

3.7.2 Classi

3.7.2.1 Premi::Back-End::App::Models::NodeContentModel

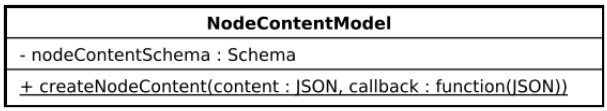


Figura 87: Premi::Back-End::App::Models::NodeContentModel

- **Descrizione**
  Questa classe rappresenta gli elementi contenuti in un nodo della mappa mentale. Contiene le informazioni riguardanti la posizione e le dimensioni dell'oggetto all'interno del nodo.
- **Utilizzo**
  Viene utilizzata per rappresentare gli elementi contenuti in un nodo, distinguendo fra titolo, contenuto testuale e URL di un'immagine. Si interfaccia alla libreria Mongoose per la creazione dello schema e dei relativi metodi statici o di istanza.
- **Relazioni con altre classi**:
  - *IN* `NodeModel`
    Questa classe rappresenta i nodi della mappa mentale.
- **Attributi**:
  - `- nodeContentSchema: Schema`
    Questo campo dati rappresenta lo schema Mongoose per il contenuto di un nodo e prevede i seguenti attributi:
    * `content` di tipo `String`, rappresenta il contenuto di un elemento;
    * `x` di tipo `Number`, rappresenta la posizione sull'asse x di un elemento nel frame;

    * `y` di tipo `Number`, rappresenta la posizione sull'asse y di un elemento nel frame;
    * `height` di tipo `Number`, rappresenta l'altezza di un elemento nel frame;
    * `width` di tipo `Number`, rappresenta la larghezza di un elemento nel frame;
    * `class` di tipo `enum`, con tre possibili valori:
      · `title`;
      · `text`;
      · `imgURL`;

      Questo attributo permette di capire il ruolo dell'elemento nel frame.

- **Metodi**:
  - + `createNodeContent(content: JSON, callback: function(JSON))`
    Metodo statico che costruisce un nuovo elemento di contenuto. Restituisce un oggetto JSON che descrive l'elemento.
    Parametri:
    * `content: JSON`
      Rappresenta i dati del contenuto da utilizzare nella creazione.
    * `callback: function(JSON)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.

3.7.2.2 Premi::Back-End::App::Models::NodeModel

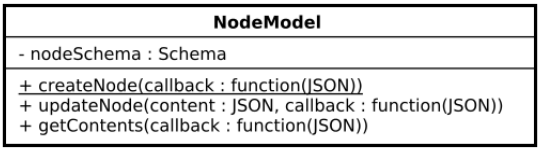


Figura 88: Premi::Back-End::App::Models::NodeModel

- **Descrizione**
  Questa classe rappresenta i nodi della mappa mentale.
- **Utilizzo**
  Viene utilizzata per rappresentare i nodi di una mappa mentale. Si interfaccia alla libreria Mongoose per la creazione dello schema e dei relativi metodi statici o di istanza.
- **Relazioni con altre classi**:
  - *IN* `PathModel`
    Questa classe rappresenta i percorsi di presentazione eseguibili su una mappa mentale. Ogni percorso è un aggregato dei nodi che l'utente vuole visualizzare nella presentazione.
  - *IN* `ProjectModel`
    Questa classe rappresenta una mappa mentale, che contiene nodi, relazioni tra i nodi e percorsi di presentazione.
  - *IN* `RelationModel`
    Questa classe rappresenta le relazioni tra i nodi di una mappa mentale. Ogni oggetto di tipo `RelationModel` modella il collegamento tra due nodi, di cui uno sarà individuato come sorgente della relazione e l'altro come destinazione. Le relazioni possono essere di due tipi, gerarchica (quando si vuole indicare che un nodo è figlio di un altro, come per un albero) oppure associazione.

- *OUT* `NodeContentModel`
  Questa classe rappresenta gli elementi contenuti in un nodo della mappa mentale. Contiene le informazioni riguardanti la posizione e le dimensioni dell'oggetto all'interno del nodo.

- **Attributi**:
  - `- nodeSchema: Schema`
    Questo campo dati rappresenta lo schema Mongoose per i nodi e prevede il seguente attributo:
    * `content` di tipo `Array`, contiene oggetti di tipo `NodeContent`, che modellano i contenuti di un nodo. Essi devono essere trattati come subdocuments per Mongoose.

- **Metodi**:
  - `+ getContents(callback: function(JSON))`
    Questo metodo restituisce un array di oggetti JSON che rappresentano i contenuti di un nodo.
    Parametri:
    * `callback: function(JSON)`
      Rappresenta la $callback_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
  - `+ updateNode(content: JSON, callback: function(JSON))`
    Questo metodo aggiorna i contenuti di un nodo. Restituisce un oggetto JSON che descrive l'elemento dopo l'aggiornamento oppure un messaggio di errore.
    Parametri:
    * `content: JSON`
      Rappresenta un array di contenuti da utilizzare per aggiornare i contenuti di un nodo.
    * `callback: function(JSON)`
      Rappresenta la $callback_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
  - `+ createNode(content: JSON, callback: function(JSON))`
    Metodo statico che costruisce un nuovo nodo senza contenuti. Restituisce un oggetto JSON che descrive l'elemento oppure un messaggio di errore.
    Parametri:
    * `content: JSON`
      Rappresenta i dati del contenuto da utilizzare nella creazione.
    * `callback: function(JSON)`
      Rappresenta la $callback_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.

3.7.2.3 Premi::Back-End::App::Models::PathModel

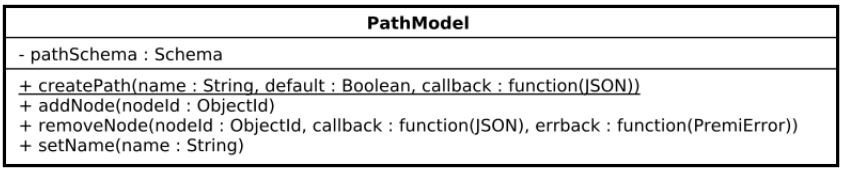


Figura 89: Premi::Back-End::App::Models::PathModel

- **Descrizione**
  Questa classe rappresenta i percorsi di presentazione eseguibili su una mappa mentale. Ogni percorso è un aggregato dei nodi che l'utente vuole visualizzare nella presentazione.
- **Utilizzo**
  Viene utilizzata per rappresentare i percorsi di presentazione creabili ed eseguibili a partire da una mappa mentale. Si interfaccia alla libreria Mongoose per la creazione dello schema e dei relativi metodi statici o di istanza. Ogni progetto ha sempre un percorso di presentazione di default, che comprende tutti i nodi presenti nella mappa mentale, ordinati secondo l'ordine di creazione.
- **Relazioni con altre classi**:
  - *IN* `ProjectModel`
    Questa classe rappresenta una mappa mentale, che contiene nodi, relazioni tra i nodi e percorsi di presentazione.
  - *OUT* `NodeModel`
    Questa classe rappresenta i nodi della mappa mentale.
- **Attributi**:
  - `- pathSchema: Schema`
    Questo campo dati rappresenta lo schema Mongoose per i percorsi di presentazione e prevede i seguenti attributi:
    * `nodes` di tipo `Array`, contiene gli `ObjectId` dei nodi presenti in un percorso di presentazione;
    * `name` di tipo `String`, rappresenta il nome che l'utente ha scelto per il percorso di presentazione;
    * `default` di tipo `Boolean`, indica se il percorso è quello di default per il progetto.
- **Metodi**:
  - `+ addNode(nodeId: ObjectId)`
    Metodo che consente di aggiungere nodi al percorso di presentazione.
    Parametri:
    * `nodeId: ObjectId`
      Rappresenta l'identificativo del nodo nel database da aggiungere al percorso di presentazione.
  - `+ removeNode(nodeId: ObjectId, callback: function(JSON), errback: function(PremiError))`
    Metodo che consente di rimuovere nodi dal percorso di presentazione. Restituisce un oggetto JSON che descrive l'elemento successivamente alla rimozione del nodo oppure un messaggio di errore.
    Parametri:
    * `nodeId: ObjectId`
      Rappresenta l'identificativo del nodo nel database da rimuovere dal percorso di presentazione.
    * `callback: function(JSON)`
      Rappresenta la $callback_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
    * `errback: function(PremiError)`
      Rappresenta la $callback_G$ che il metodo deve chiamare qualora si verificassero errori durante l'esecuzione del metodo.
  - `+ setName(name: String)`
    Metodo che consente di modificare il nome del percorso di presentazione.
    Parametri:

  * `name: String`
    Rappresenta il nuovo nome del percorso di presentazione.

- + `createPath(name: String, default: Boolean, callback: function(JSON))`
  Metodo statico che costruisce un nuovo percorso inizialmente privo di nodi. Restituisce un oggetto JSON che descrive l'elemento.
  Parametri:
  * `name: String`
    Rappresenta il nome del percorso che si vuole creare.
  * `default: Boolean`
    Specifica se il percorso che da creare è quello di default.
  * `callback: function(JSON)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.

3.7.2.4 Premi::Back-End::App::Models::ProjectModel

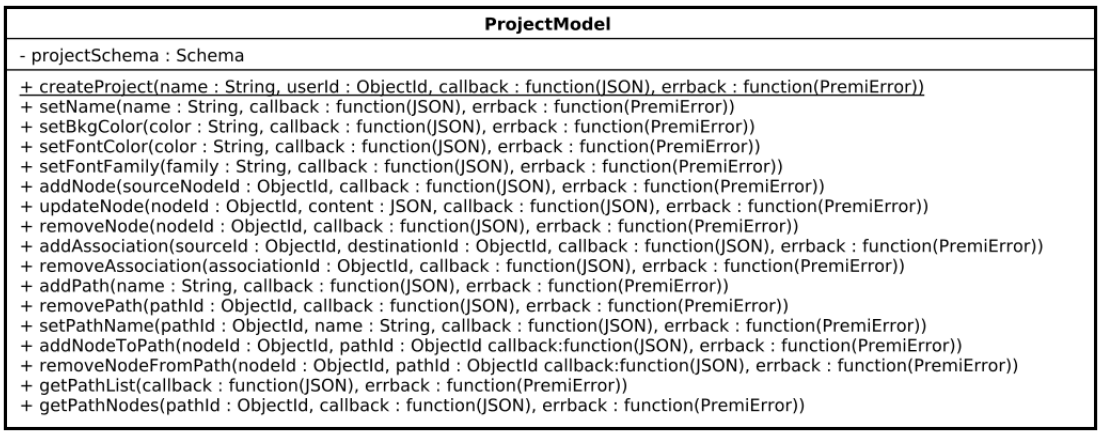


Figura 90: Premi::Back-End::App::Models::ProjectModel

- **Descrizione**
  Questa classe rappresenta una mappa mentale, che contiene nodi, relazioni tra i nodi e percorsi di presentazione.
- **Utilizzo**
  Viene utilizzata per rappresentare una mappa mentale. Si interfaccia alla libreria Mongoose per la creazione dello schema e dei relativi metodi statici o di istanza. Questa classe implementa il design pattern Façade e consente ai controller di interagire con le classi del model che realizzano il progetto attraverso la sua interfaccia pubblica.
- **Relazioni con altre classi**:
  - *IN* `NodeController`
    Classe che gestisce la logica applicativa riguardante la visualizzazione e la modifica dei nodi presenti in un progetto.
  - *IN* `PathController`
    Classe che gestisce la logica applicativa riguardante la visualizzazione e la modifica dei percorsi di presentazione di un progetto.

- *IN* `ProjectManagementController`
  Classe che gestisce la logica applicativa riguardante la visualizzazione e la modifica dei progetti. Rappresenta il ConcreteHandler nel design pattern Chain of responsibility. Utilizza Passport.
- *IN* `UserModel`
  Classe che modella la creazione e la gestione dei dati utente.
- *OUT* `NodeModel`
  Questa classe rappresenta i nodi della mappa mentale.
- *OUT* `PathModel`
  Questa classe rappresenta i percorsi di presentazione eseguibili su una mappa mentale. Ogni percorso è un aggregato dei nodi che l'utente vuole visualizzare nella presentazione.
- *OUT* `RelationModel`
  Questa classe rappresenta le relazioni tra i nodi di una mappa mentale. Ogni oggetto di tipo `RelationModel` modella il collegamento tra due nodi, di cui uno sarà individuato come sorgente della relazione e l'altro come destinazione. Le relazioni possono essere di due tipi, gerarchica (quando si vuole indicare che un nodo è figlio di un altro, come per un albero) oppure associazione.

- **Attributi**:
  - `- projectSchema: Schema`
    Questo campo dati rappresenta lo schema Mongoose per i progetti e prevede i seguenti attributi:
    * `nodes` di tipo `Array`, contiene oggetti di tipo `Node` che devono essere trattati come subdocuments per Mongoose
    * `root` di tipo `ObjectId`, rappresenta il riferimento all'identificativo del nodo radice della mappa mentale;
    * `relations` di tipo `Array`, contiene oggetti di tipo `Relation` che devono essere trattati come subdocuments per Mongoose;
    * `paths` di tipo `Array`, contiene oggetti di tipo `Path` che devono essere trattati come subdocuments per Mongoose;
    * `name` di tipo `String`, rappresenta il nome che l'utente ha scelto per il progetto;
    * `userId` di tipo `ObjectId`, rappresenta il riferimento all'identificativo nel database dell'utente proprietario del progetto;
    * `bkgColor` di tipo `String`, rappresenta il colore scelto per lo sfondo dei frame in presentazione;
    * `fontColor` di tipo `String`, rappresenta il colore scelto per i testi nei frame in presentazione;
    * `fontFamily` di tipo `String`, rappresenta la famiglia di font scelta per i testi nei frame in presentazione.

- **Metodi**:
  - `+ addAssociation(sourceId: ObjectId, destinationId: ObjectId, callback: function(JSON), errback: function(PremiError)): JSON`
    Metodo che costruisce una nuova relazione di tipo associazione. Restituisce un oggetto JSON che descrive l'elemento oppure un messaggio di errore.
    Parametri:
    * `sourceId: ObjectId`
      Rappresenta l'identificativo nel database del nodo sorgente della relazione.

    * `destinationId: ObjectId`
      Rappresenta l'identificativo nel database del nodo destinazione della relazione.
    * `callback: function(JSON)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
    * `errback: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare qualora si verificassero errori durante l'esecuzione del metodo.

- `+ addNode(sourceNodeId: ObjectId, callback: function(JSON), errback: function(PremiError)): JSON`
  Metodo che consente di aggiungere nodi vuoti al progetto e quindi alla mappa mentale. Crea inoltre la relazione gerarchica tra il nuovo nodo e quello specificato come sorgente e lo aggiunge al percorso di default. Restituisce un oggetto JSON che descrive l'elemento aggiunto oppure un messaggio di errore.
  Parametri:
    * `sourceNodeId: ObjectId`
      Rappresenta l'identificativo del nodo sorgente della relazione nel database.
    * `callback: function(JSON)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
    * `errback: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare qualora si verificassero errori durante l'esecuzione del metodo.

- `+ addNodeToPath(nodeId: ObjectId, pathId: ObjectId, callback: function(JSON), errback: function(PremiError)): JSON`
  Metodo che consente di aggiungere nodi ad un percorso di presentazione. Restituisce un oggetto JSON che descrive l'elemento oppure un messaggio di errore.
  Parametri:
    * `nodeId: ObjectId`
      Rappresenta l'identificativo del nodo nel database, da aggiungere al percorso di presentazione.
    * `pathId: ObjectId`
      Rappresenta l'identificativo del percorso nel database a cui aggiungere il nodo.
    * `callback: function(JSON)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
    * `errback: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare qualora si verificassero errori durante l'esecuzione del metodo.

- `+ addPath(pathId: ObjectId, callback: function(JSON), errback: function(PremiError)): JSON`
  Metodo che consente di aggiungere un nuovo percorso di presentazione con un nome specificato nei parametri. Restituisce un oggetto JSON che descrive l'elemento oppure un messaggio di errore.
  Parametri:
    * `pathId: ObjectId`
      Rappresenta l'identificativo del percorso da rimuovere.
    * `callback: function(JSON)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.

    * `errback: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare qualora si verificassero errori durante l'esecuzione del metodo.

- `+ getPathList(callback: function(JSON), errback: function(PremiError)): JSON`
  Metodo che consente di ottenere l'elenco dei percorsi di presentazione del progetto, ovvero un oggetto JSON contenente coppie di identificativi e nomi dei percorsi, attraverso la funzione di callback.
  Parametri:
    * `callback: function(JSON)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
    * `errback: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare qualora si verificassero errori durante l'esecuzione del metodo.

- `+ getPathNodes(pathId: ObjectId, callback: function(JSON), errback: function(PremiError)): JSON`
  Metodo che consente di ottenere l'elenco dei nodi presenti in un percorso di presentazione del progetto attraverso la funzione di callback.
  Parametri:
    * `pathId: ObjectId`
      Rappresenta l'identificativo nel database del percorso di cui si vogliono ottenere i nodi.
    * `callback: function(JSON)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
    * `errback: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare qualora si verificassero errori durante l'esecuzione del metodo.

- `+ removeAssociation(callback: function(JSON), errback: function(PremiError), associationId: ObjectId): JSON`
  Metodo statico che rimuove un relazione di tipo `association`. Restituisce un oggetto JSON che descrive l'elemento rimosso oppure un messaggio di errore.
  Parametri:
    * `callback: function(JSON)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
    * `errback: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare qualora si verificassero errori durante l'esecuzione del metodo.
    * `associationId: ObjectId`
      Rappresenta l'identificativo nel database dell'associazione da rimuovere.

- `+ removeNode(nodeId: ObjectId, callback: function(JSON), errback: function(PremiError)): JSON`
  Metodo che consente di rimuovere nodi dal progetto e quindi dalla mappa mentale. Se un nodo da rimuovere è radice di un sottoalbero, devono essere rimossi anche tutti i figli e le relazioni che coinvolgono il nodo. Deve aggiornare anche tutti i percorsi di presentazione che contenevano quel nodo. Restituisce un oggetto JSON che descrive la mappa mentale priva del sottoalbero rimosso oppure un messaggio di errore.
  Parametri:
    * `nodeId: ObjectId`
      Rappresenta l'identificativo del nodo nel database, da rimuovere.

    * `callback: function(JSON)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
    * `errback: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare qualora si verificassero errori durante l'esecuzione del metodo.

- `+ removeNodeFromPath(nodeId: ObjectId, pathId: ObjectId, callback: function, errback: function(PremiError)): JSON`
  Metodo che consente di rimuovere nodi da un percorso di presentazione del progetto. Restituisce un oggetto JSON che descrive l'elemento oppure un messaggio di errore.
  Parametri:
    * `nodeId: ObjectId`
      Questo parametro rappresenta l'identificativo del nodo nel database, da rimuovere dal percorso di presentazione.
    * `pathId: ObjectId`
      Questo attributo rappresenta l'identificativo del percorso nel database in cui rimuovere il nodo.
    * `callback: function`
      Questo parametro rappresenta la callback che il metodo deve chiamare in caso di terminazione senza errori.
    * `errback: function(PremiError)`
      Questo parametro rappresenta la callback che il metodo deve chiamare qualora si verificassero errori nell'esecuzione del metodo.

- `+ removePath(name: String, callback: function(JSON), errback: function(PremiError)): JSON`
  Metodo che consente di rimuovere un percorso di presentazione. Restituisce un oggetto JSON che descrive l'elemento oppure un messaggio di errore.
  Parametri:
    * `name: String`
      Rappresenta il nome del percorso da rimuovere.
    * `callback: function(JSON)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
    * `errback: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare qualora si verificassero errori durante l'esecuzione del metodo.

- `+ setBkgColor(color: String, callback: function(JSON), errback: function(PremiError)): JSON`
  Metodo che consente di modificare il colore dello sfondo dei frame in presentazione del progetto. Restituisce un oggetto JSON che descrive l'elemento oppure un messaggio di errore.
  Parametri:
    * `color: String`
      Rappresenta il nome del colore di sfondo da impostare.
    * `callback: function(JSON)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
    * `errback: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare qualora si verificassero errori durante l'esecuzione del metodo.

- `+ setFontColor(color: String, callback: function(JSON), errback: function(PremiError)): JSON`
  Metodo che consente di modificare il colore dei testi dei frame in presentazione del progetto. Restituisce un oggetto JSON che descrive l'elemento oppure un messaggio di errore.
  Parametri:
  * `color: String`
    Rappresenta il colore da impostare come colore del testo.
  * `callback: function(JSON)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
  * `errback: function(PremiError)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare qualora si verificassero errori durante l'esecuzione del metodo.
- `+ setFontFamily(family: String, callback: function(JSON), errback: function(PremiError)): JSON`
  Metodo che consente di modificare la famiglia di font dei testi dei frame in presentazione del progetto. Restituisce un oggetto JSON che descrive l'elemento oppure un messaggio di errore.
  Parametri:
  * `family: String`
    Rappresenta il nome della nuova famiglia di font per i testi del progetto.
  * `callback: function(JSON)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
  * `errback: function(PremiError)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare qualora si verificassero errori durante l'esecuzione del metodo.
- `+ setName(name: String, callback: function(JSON), errback: function(PremiError)): JSON`
  Metodo che consente di modificare il nome del progetto. Restituisce un oggetto JSON che descrive l'elemento oppure un messaggio di errore.
  Parametri:
  * `name: String`
    Rappresenta il nuovo nome del progetto.
  * `callback: function(JSON)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
  * `errback: function(PremiError)`
    Rappresenta la *callback*$_G$ che il metodo deve chiamare qualora si verificassero errori durante l'esecuzione del metodo.
- `+ setPathName(name: String, pathId: ObjectId, callback: function(JSON), errback: function(PremiError)): JSON`
  Metodo che consente di modificare il nome di un percorso di presentazione. Restituisce un oggetto JSON che descrive l'elemento oppure un messaggio di errore.
  Parametri:
  * `name: String`
    Rappresenta il nuovo nome del percorso di presentazione.
  * `pathId: ObjectId`
    Rappresenta l'identificativo nel database del percorso di cui si vuole modificare il nome.

    * `callback: function(JSON)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
    * `errback: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare qualora si verificassero errori durante l'esecuzione del metodo.
- `+ updateNode(nodeId: ObjectId, content: JSON, callback: function(JSON), errback: function(PremiError)): JSON`
  Metodo che consente di modificare il contenuto di un nodo nella mappa mentale.
  Parametri:
    * `nodeId: ObjectId`
      Rappresenta l'identificativo del nodo da modificare.
    * `content: JSON`
      Rappresenta un array di oggetti JSON con cui aggiornare il contenuto di un nodo. I contenuti precedenti vengono sovrascritti.
    * `callback: function(JSON)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
    * `errback: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare qualora si verificassero errori durante l'esecuzione del metodo.
- `+ createProject(name: String, userId: ObjectId, callback: function(JSON), errback: function(PremiError)): JSON`
  Metodo statico che costruisce un nuovo progetto che conterrà il primo nodo vuoto della mappa mentale. Verrà inoltre istanziato il percorso di default. Restituisce un oggetto JSON che descrive l'elemento oppure un messaggio di errore.
  Parametri:
    * `name: String`
      Rappresenta il nome del progetto che si vuole creare.
    * `userId: ObjectId`
      Rappresenta l'id dell'utente che ha creato il progetto.
    * `callback: function(JSON)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
    * `errback: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare qualora si verificassero errori nella costruzione di un oggetto.

3.7.2.5 Premi::Back-End::App::Models::RelationModel

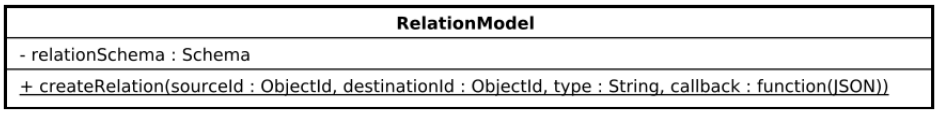


Figura 91: Premi::Back-End::App::Models::RelationModel

- **Descrizione**
  Questa classe rappresenta le relazioni tra i nodi di una mappa mentale. Ogni oggetto di tipo `RelationModel` modella il collegamento tra due nodi, di cui uno sarà individuato come

sorgente della relazione e l'altro come destinazione. Le relazioni possono essere di due tipi, gerarchica (quando si vuole indicare che un nodo è figlio di un altro, come per un albero) oppure associazione.

- **Utilizzo**
  Viene utilizzata per rappresentare le relazione tra i nodi di una mappa mentale. Si interfaccia alla libreria Mongoose per la creazione dello schema e dei relativi metodi statici o di istanza.
- **Relazioni con altre classi**:
  - *IN* `ProjectModel`
    Questa classe rappresenta una mappa mentale, che contiene nodi, relazioni tra i nodi e percorsi di presentazione.
  - *OUT* `NodeModel`
    Questa classe rappresenta i nodi della mappa mentale.
- **Attributi**:
  - `- relationSchema: Schema`
    Questo campo dati rappresenta lo schema Mongoose per le relazioni e prevede i seguenti attributi:
    * `source` di tipo `ObjectId`, rappresenta l'identificativo nel database del nodo sorgente della relazione;
    * `destination` di tipo `ObjectId`, rappresenta l'identificativo nel database del nodo destinazione della relazione;
    * `class` di tipo enum, con due possibili valori:
      · `hierarchical`;
      · `association`.
      Essi consentono di distinguere tra relazioni gerarchiche e associazioni.
- **Metodi**:
  - `+ <u>createRelation</u>(sourceId: ObjectId, destinationId: ObjectId, type: String, callback: function(JSON), errback: function(PremiError))`
    Metodo statico che costruisce un nuova relazione. Restituisce un oggetto JSON che descrive l'elemento oppure un messaggio di errore.
    Parametri:
    * `sourceId: ObjectId`
      Rappresenta l'identificativo nel database del nodo sorgente della relazione.
    * `destinationId: ObjectId`
      Rappresenta l'identificativo nel database del nodo destinazione della relazione.
    * `type: String`
      Rappresenta il tipo di relazione che deve essere creata, scegliendo tra relazione gerarchica o associazione.
    * `callback: function(JSON)`
      Rappresenta la $callback_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
    * `errback: function(PremiError)`
      Rappresenta la $callback_G$ che il metodo deve chiamare qualora si verificassero errori nella costruzione di un oggetto.

3.7.2.6 Premi::Back-End::App::Models::Session

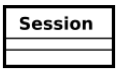


Figura 92: Premi::Back-End::App::Models::Session

- **Descrizione**
  Classe che gestisce la sessione utente dell'applicazione. *Non sono stati modellati attributi e metodi di questa classe in quanto viene inizializzata da Express ed utilizzata da Passport attraverso funzionalità interne ai due middleware.*
- **Utilizzo**
  Viene utilizzata da Passport per memorizzare i dati della sessione che viene creata quando un utente effettua il login.
- **Relazioni con altre classi**:
  - *IN* `AuthenticationController`
    Classe che si occupa della registrazione e dell'autenticazione dell'utente nel sistema. È un componente ConcreteHandler del design pattern Chain of responsibility. Risulta essere il componente che eventualmente esegue la richiesta del client attraverso Passport.
  - *IN* `AuthorizationController`
    Classe middleware che, utilizzando Passport, si occupa di controllare la consistenza dell'oggetto `session` durante la sessione associata all'utente autenticato. È un componente ConcreteHandler del design pattern Chain of responsibility.

3.7.2.7 Premi::Back-End::App::Models::UserModel

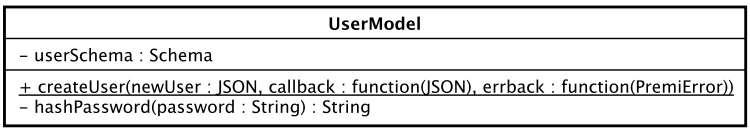


Figura 93: Premi::Back-End::App::Models::UserModel

- **Descrizione**
  Classe che modella la creazione e la gestione dei dati utente.
- **Utilizzo**
  Viene utilizzata per rappresentare i dati degli account dei vari utenti dell'applicazione. Si interfaccia alla libreria Mongoose per la creazione dello schema e dei relativi metodi statici o di istanza.
- **Relazioni con altre classi**:
  - *IN* `AuthenticationController`
    Classe che si occupa della registrazione e dell'autenticazione dell'utente nel sistema. È un componente ConcreteHandler del design pattern Chain of responsibility. Risulta essere il componente che eventualmente esegue la richiesta del client attraverso Passport.
  - *IN* `AuthorizationController`
    Classe middleware che, utilizzando Passport, si occupa di controllare la consistenza dell'oggetto `session` durante la sessione associata all'utente autenticato. È un componente ConcreteHandler del design pattern Chain of responsibility.

- *OUT* `ProjectModel`
  Questa classe rappresenta una mappa mentale, che contiene nodi, relazioni tra i nodi e percorsi di presentazione.

- **Attributi**:
  - `- userSchema: Schema`
    Questo campo dati rappresenta lo schema Mongoose dell'utente Premi. Lo schema prevede i seguenti attributi:
    * `username` di tipo `String`, rappresenta lo username con cui viene identificato l'utente all'interno dell'applicazione;
    * `password` di tipo `String`, rappresenta la password associata all'utente, viene dapprima concatenata con un campo dati salt generato internamente da Passport e successivamente viene eseguito l'hashing;

- **Metodi**:
  - `- hashPassword(password: String): String`
    Effettua l'hashing della stringa `password` se non è già stata criptata tramite campo `salt` per evitare attacchi di tipo *rainbow*.
    Parametri:
    * `password: String`
      Rappresenta la password dell'utente.
  - `+ createUser(newUser: JSON, callback: function(JSON), errback: function(PremiError))`
    Metodo statico che costruisce un nuovo account associato ad un indirizzo e-mail. Restituisce un oggetto JSON che descrive l'elemento inserito oppure ritorna un messaggio di errore.
    Parametri:
    * `newUser: JSON`
      Rappresenta i dati da utilizzare per la creazione di un nuovo utente.
    * `callback: function(JSON)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare al termine dell'elaborazione nel caso in cui non si siano verificati errori.
    * `errback: function(PremiError)`
      Rappresenta la *callback*$_G$ che il metodo deve chiamare qualora si verificassero errori nella costruzione di un oggetto.

3.8 Premi::Back-End::App::Routers

3.8.1 Informazioni generali

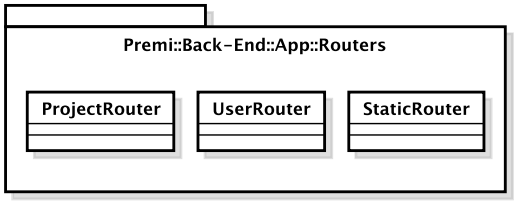


Figura 94: Premi::Back-End::App::Routers

- **Descrizione**
  Package contenente i router della componente back-end dell'applicazione. Contiene i file di configurazione relativi al routing delle richieste del client, ossia i routes di Express.
- **Padre**: `App`
- **Interazioni con altri componenti**:
  - `Controllers`
    Package che contiene i controllers di Express, definisce la logica dell'applicazione.

3.8.2 Classi

3.8.2.1 Premi::Back-End::App::Routers::ProjectRouter

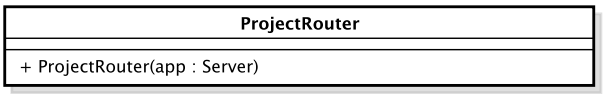


Figura 95: Premi::Back-End::App::Routers::ProjectRouter

- **Descrizione**
  Classe che gestisce le richieste relative alle operazioni riguardanti la mappa. Componente ConcreteHandler del design pattern Chain of responsibility.
- **Utilizzo**
  Viene utilizzata per chiamare il controller che si occupa di gestire le API relative alla mappa.
- **Relazioni con altre classi**:
  - *IN* `Server`
    Classe che avvia il server. Nello specifico apre una connessione al database tramite Mongoose, invoca il middleware Express passando un riferimento al database MongoDB come parametro in modo che possa configurarsi con esso, invoca il middleware Passport ed infine si mette in ascolto su una determinata porta. È il componente client del design pattern Chain of responsibility. Utilizza i moduli Mongoose, Express, Passport e Chalk, quest'ultimo serve per definire lo stile dei tipi `String`.
  - *OUT* `ErrorHandler`
    Classe middleware per la gestione degli errori. Ritorna al client un oggetto di tipo `Response` con stato HTTP 500 e descrizione dell'errore in formato JSON. È un componente ConcreteHandler del design pattern Chain of responsibility.
  - *OUT* `NotFoundHandler`
    Classe che si occupa della gestione dell'errore di pagina non trovata. Componente ConcreteHandler del design pattern Chain of responsibility.
  - *OUT* `ProjectController`
    Classe che raggruppa i vari controllers responsabili delle operazioni riguardanti la mappa mentale attraverso `require`.
- **Metodi**:
  - `+ ProjectRouter(app: Server)`
    Contiene diverse route che vengono configurate all'avvio del server. Quest'ultime ricevono le richieste del client e passano il controllo al ConcreteHandler successivo.
    Parametri:

* `app: Server`
  Rappresenta l'istanza del server su cui configurare i route che mappano i controllers specifici.

3.8.2.2 Premi::Back-End::App::Routers::StaticRouter

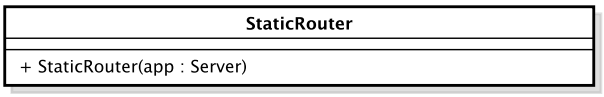


Figura 96: Premi::Back-End::App::Routers::StaticRouter

- **Descrizione**
  Classe che gestisce le richieste relative alle pagine statiche della componente back-end. Componente ConcreteHandler del design pattern Chain of responsibility.
- **Utilizzo**
  Viene utilizzata per fornire delle pagine HTML statiche in risposta al client, in particolare fornisce il manuale utente online.
- **Relazioni con altre classi**:
  - *IN* `Server`
    Classe che avvia il server. Nello specifico apre una connessione al database tramite Mongoose, invoca il middleware Express passando un riferimento al database MongoDB come parametro in modo che possa configurarsi con esso, invoca il middleware Passport ed infine si mette in ascolto su una determinata porta. È il componente client del design pattern Chain of responsibility. Utilizza i moduli Mongoose, Express, Passport e Chalk, quest'ultimo serve per definire lo stile dei tipi `String`.
  - *OUT* `ErrorHandler`
    Classe middleware per la gestione degli errori. Ritorna al client un oggetto di tipo `Response` con stato HTTP 500 e descrizione dell'errore in formato JSON. È un componente ConcreteHandler del design pattern Chain of responsibility.
  - *OUT* `NotFoundHandler`
    Classe che si occupa della gestione dell'errore di pagina non trovata. Componente ConcreteHandler del design pattern Chain of responsibility.
  - *OUT* `StaticController`
    Classe che gestisce le operazioni e la logica applicativa riguardante la visualizzazione di pagine HTML statiche. Rappresenta uno dei componenti ConcreteHandler del design pattern Chain of responsibility.
- **Metodi**:
  - `+ StaticRouter(app: Server)`
    Contiene diverse route che vengono configurate all'avvio del server. Quest'ultime ricevono le richieste del client e passano il controllo al ConcreteHandler successivo.
    Parametri:
    * `app: Server`
      Rappresenta l'istanza del server su cui configurare i route che mappano i controllers specifici.

3.8.2.3 Premi::Back-End::App::Routers::UserRouter

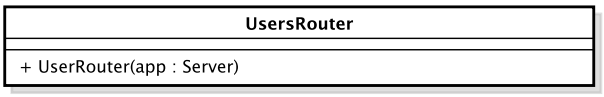


Figura 97: Premi::Back-End::App::Routers::UserRouter

- **Descrizione**
  Classe che gestisce le richieste relative alla registrazione e alla gestione della sessione di un utente. Componente ConcreteHandler del design pattern Chain of responsibility. Utilizza il modulo Passport.
- **Utilizzo**
  Viene utilizzata per chiamare il controller che si occupa di gestire le API relative alla registrazione e alla gestione della sessione di un utente.
- **Relazioni con altre classi**:
  - *IN* `Server`
    Classe che avvia il server. Nello specifico apre una connessione al database tramite Mongoose, invoca il middleware Express passando un riferimento al database MongoDB come parametro in modo che possa configurarsi con esso, invoca il middleware Passport ed infine si mette in ascolto su una determinata porta. È il componente client del design pattern Chain of responsibility. Utilizza i moduli Mongoose, Express, Passport e Chalk, quest'ultimo serve per definire lo stile dei tipi `String`.
  - *OUT* `ErrorHandler`
    Classe middleware per la gestione degli errori. Ritorna al client un oggetto di tipo `Response` con stato HTTP 500 e descrizione dell'errore in formato JSON. È un componente ConcreteHandler del design pattern Chain of responsibility.
  - *OUT* `NotFoundHandler`
    Classe che si occupa della gestione dell'errore di pagina non trovata. Componente ConcreteHandler del design pattern Chain of responsibility.
  - *OUT* `UserController`
    Classe che raggruppa attraverso `require` i vari controllers responsabili delle operazioni legate alla gestione degli utenti. Si è scelto di predisporre questo raggruppamento per facilitare l'introduzione di nuove funzionalità legate alla gestione degli utenti.
- **Metodi**:
  - `+ UserRouter(app: Server)`
    Contiene diverse route che vengono configurate all'avvio del server. Quest'ultime ricevono le richieste del client e passano il controllo al ConcreteHandler successivo.
    Parametri:
    * `app: Server`
      Rappresenta l'istanza del server su cui configurare i route che mappano i controllers specifici.

3.9 Premi::Back-End::App::Views

3.9.1 Informazioni generali

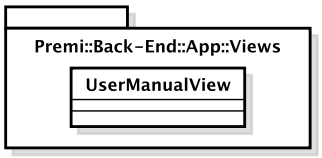


Figura 98: Premi::Back-End::App::Views

- **Descrizione**
  Package contenente le views della componente back-end dell'applicazione.
- **Padre**: `App`
- **Interazioni con altri componenti**:
  - `Controllers`
    Package che contiene i controllers di Express, definisce la logica dell'applicazione.

3.9.2 Classi

3.9.2.1 Premi::Back-End::App::Views::UserManualView

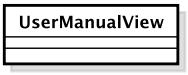


Figura 99: Premi::Back-End::App::Views::UserManualView

- **Descrizione**
  Pagina HTML statica contenente il manuale dell'applicazione.
- **Utilizzo**
  Viene visualizzata quando l'utente richiede di consultare il manuale utente.
- **Relazioni con altre classi**:
  - *IN* `StaticController`
    Classe che gestisce le operazioni e la logica applicativa riguardante la visualizzazione di pagine HTML statiche. Rappresenta uno dei componenti ConcreteHandler del design pattern Chain of responsibility.

3.10 Premi::Back-End::Config

3.10.1 Informazioni generali

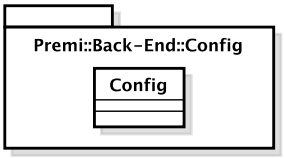


Figura 100: Premi::Back-End::Config

- **Descrizione**
  Package contenente le componenti di configurazione del server.
- **Padre**: `Back-End`
- **Interazioni con altri componenti**:
  - `App`
    Package contenente le componenti del server che implementano il pattern MVC.

3.10.2 Classi

3.10.2.1 Premi::Back-End::Config::Config

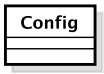


Figura 101: Premi::Back-End::Config::Config

- **Descrizione**
  Questa classe gestisce la configurazione del server. *Non sono stati modellati attributi e metodi di questa classe in quanto viene gestita da Express.*
- **Utilizzo**
  Viene utilizzata per descrivere i parametri dell'applicazione. La classe `Server` utilizza oggetti di questo tipo per creare ed avviare l'istanza del server.
- **Relazioni con altre classi**:
  - *IN* `Server`
    Classe che avvia il server. Nello specifico apre una connessione al database tramite Mongoose, invoca il middleware Express passando un riferimento al database MongoDB come parametro in modo che possa configurarsi con esso, invoca il middleware Passport ed infine si mette in ascolto su una determinata porta. È il componente client del design pattern Chain of responsibility. Utilizza i moduli Mongoose, Express, Passport e Chalk, quest'ultimo serve per definire lo stile dei tipi `String`.

4 Diagrammi di sequenza

4.1 Front-End

Nei diagrammi di sequenza dei *controller*$_G$ che utilizzano le `Promise`, le due funzioni anonime che vengono fornite all'ggetto `Promise` sono state nominate `success` e `fail` anche se i *controller*$_G$ non possiedono queste due funzioni.

4.1.1 Controllers

Vengono di seguito riportate alcune sequenze di operazioni svolte dai *controller*$_G$ che si è ritenuto utile descrivere più in dettaglio.

4.1.1.1 RegistrationController - Registrazione di un nuovo utente

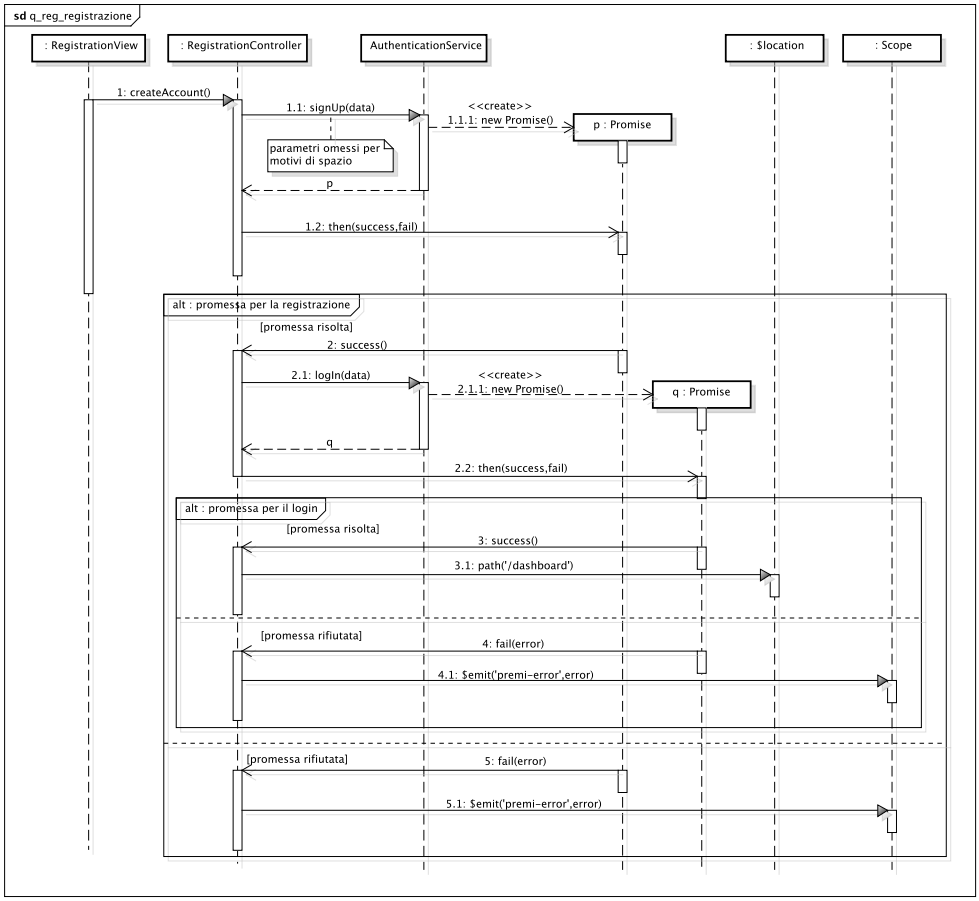


Figura 102: Diagramma di Sequenza - Registrazione di un nuovo utente

Quando l'utente preme il pulsante per la registrazione viene invocato il metodo `createAccount()`, il quale si preoccupa di andare a leggere i dati inseriti nello `$scope` dall'utente e di usarli per invocare il metodo `signUp` di `AuthenticationService`.
Il service ritorna un oggetto `Promise` al quale il *controller*$_G$ fornisce due funzioni anonime da invocare

nel caso la promessa sia risolta o rifiutata.
Se la promessa viene rifiutata, il *controller*$_G$ solleva l'evento `premi-error` per segnalare all'utente che non è stato possibile effettuare la registrazione.
Se invece la promessa viene soddisfatta, il *controller*$_G$ richiede a `AuthenticationService` l'autenticazione con gli stessi dati che l'utente ha inserito per registrarsi.
Anche in questo caso viene ritornata una promessa e, anche in questo caso, il *controller*$_G$ fornisce all'oggetto `Promise` due funzioni anonime.
Se la promessa viene risolta, il *controller*$_G$ esegue il re-indirizzamento alla dashboard altrimenti, viene sollevato l'evento `premi-error` per segnalare all'utente l'errore che si è verificato.

4.1.1.2 DashboardController - Apertura di un progetto

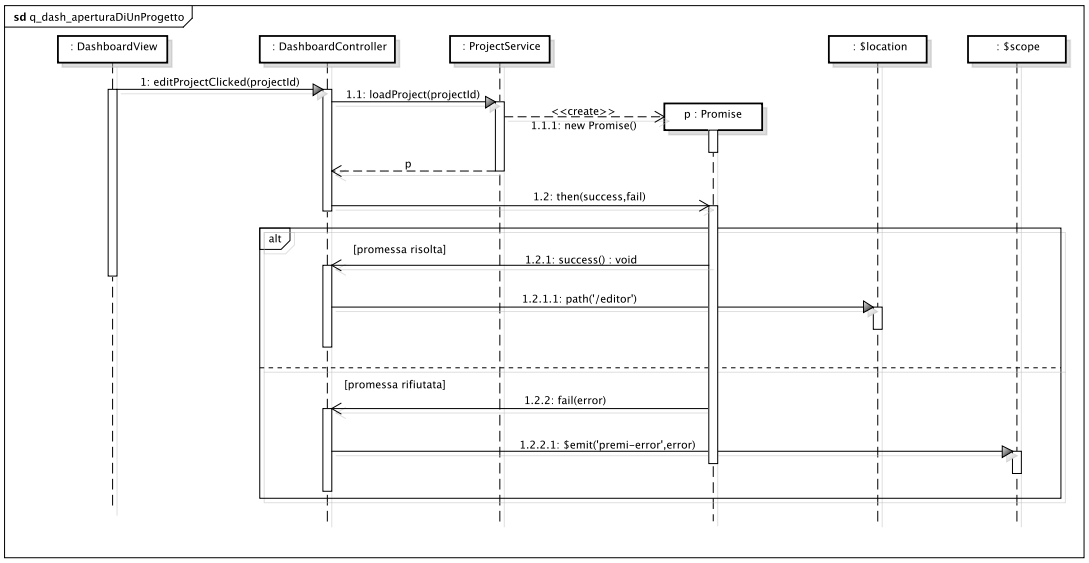


Figura 103: Diagramma di Sequenza - Apertura di un progetto

Quando l'utente preme il pulsante per la modifica di un *progetto*$_G$ viene invocato, come gestore dell'evento, il metodo `editProjectClicked(projectId)`. Questo si preoccupa di richiedere a `ProjectService` l'apertura del *progetto*$_G$ identificato da `projectId`.
La funzione del service ritorna un oggetto di tipo `Promise`, al quale il *controller*$_G$ fornisce due funzioni anonime per gestire la risoluzione o il rifiuto della promessa.
Se la promessa viene soddisfatta, il *controller*$_G$ effettua il re-indirizzamento alla pagina per l'editing del *progetto*$_G$, altrimenti, se la promessa viene rifiutata, il *controller*$_G$ segnala un'errore sollevando l'evento `premi-error`. Il re-indirizzamento porta alla costruzione di un `MindmapEditorController`, la sequenza di operazioni per la costruzione di tale *controller*$_G$ viene descritta nella sezione 4.1.1.3.

4.1.1.3 MindmapEditorController - Costruzione del controller

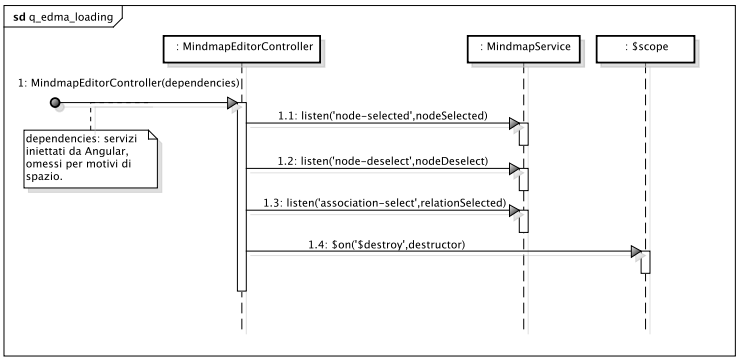


Figura 104: Diagramma di Sequenza - Costruzione del controller

Quando l'utente apre un *progetto*$_G$ e desidera visualizzarne la *mappa mentale*$_G$ per modificarla deve essere istanziato il *controller*$_G$ legato a quella *view*$_G$. In particolare viene eseguita questa sequenza di azioni:

1. Vengono registrate le varie funzioni di *callback*$_G$ per i seguenti eventi offerti da `MindmapService`:
   - `node-select`;
   - `node-deselect`;
   - `association-select`.
2. Viene registrato il distruttore del *controller*$_G$ sull'evento `$destroy` dello `$scope`.

Non è necessario recuperare e inizializzare i dati della *mappa mentale*$_G$ in quanto il caricamento dei dati viene effettuato da `ProjcetService.loadProject` e l'inizializzazione della mappa mentale è delegata alla directive `premiMindmap`.

4.1.1.4 MindmapEditorController - Selezione di un nodo

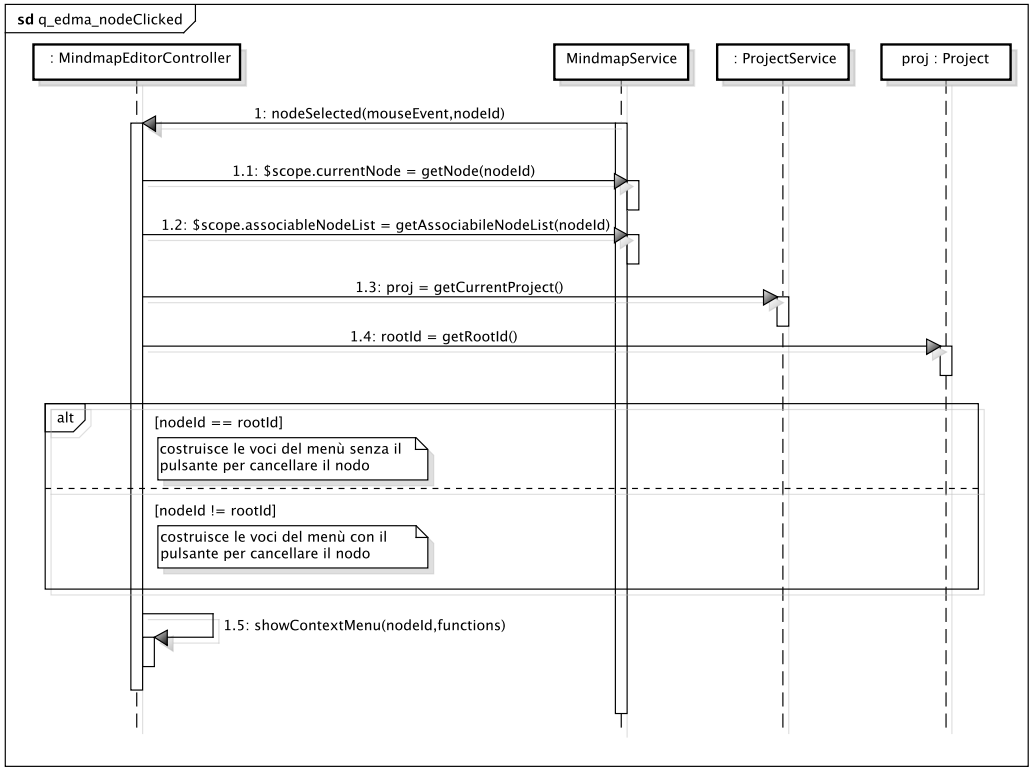


Figura 105: Diagramma di Sequenza - Selezione di un nodo

Quando l'utente seleziona un nodo della *mappa mentale*$_G$ il service `MindmapService` invoca la funzione di callback$_G$ del *controller*$_G$ in modo che l'evento possa essere gestito correttamente. L'esecuzione delle funzione `nodeSelected(mouseEvent,nodeId)` deve:

1. Aggiornare la variabile `$scope.currentNode` con il nodo identificato da `nodeId`;
2. Aggiornare la variabile `$scope.associableNodeList` con la lista dei nodi che possono essere associati al nodo corrente;
3. Recuperare le informazioni riguardanti il nodo radice del *progetto*$_G$;
4. Caricare le voci del menù e, nel caso il nodo selezionato sia la radice, disabilitare l'eliminazione del nodo;
5. Visualizzare il menù di modifica del nodo invocando la funzione `showContextMenu`.

4.1.1.5 PresentationController - Creazione di una presentazione

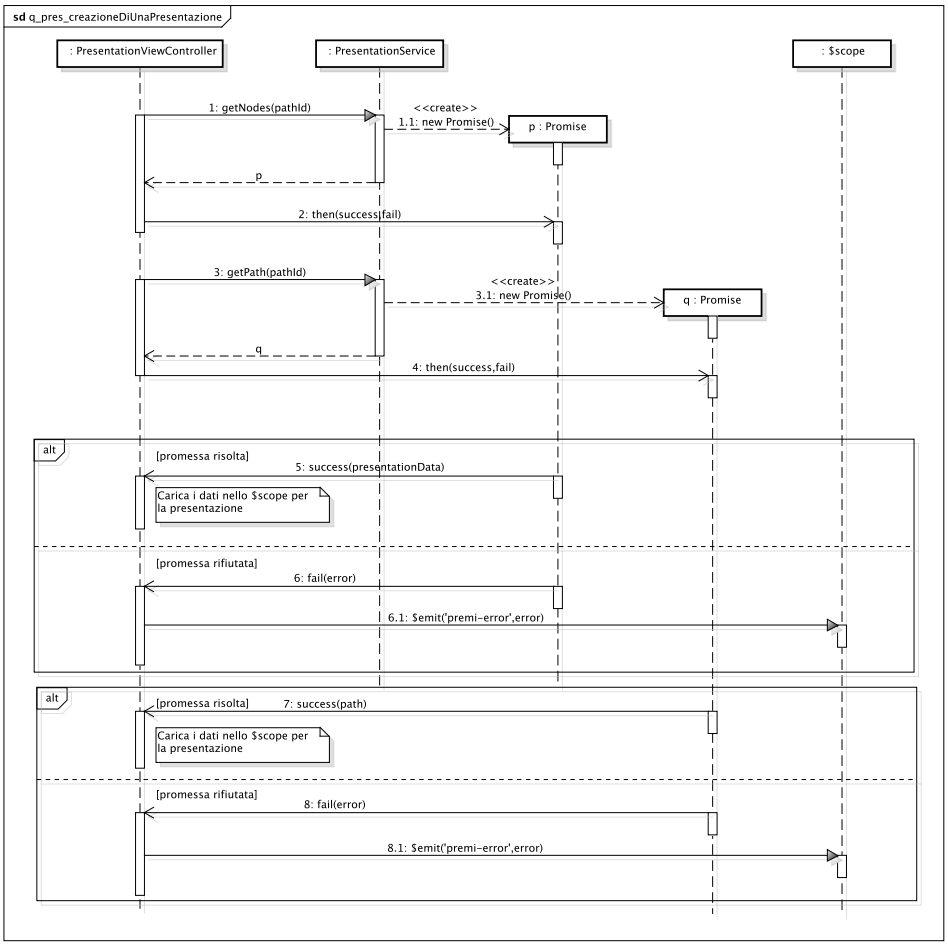


Figura 106: Diagramma di Sequenza - Creazione di una presentazione

Quando l'utente vuole visualizzare una presentazione viene indirizzato verso la `PresentationView`, la quale permette di vedere il contenuto del *frame*$_G$ dei nodi come se fosse uno slideshow.
Per far si che la presentazione venga generata correttamente, è necessario che il costruttore del *controller*$_G$ della *view*$_G$ predisponga nello `$scope` tutti gli oggetti necessari per la creazione della presentazione.
Come prima cosa il *controller*$_G$ deve recuperare l'`id` del *percorso*$_G$ da presentare, attraverso la variabile `$routeParam` fornita da *Angular*$_G$.
Una volta recuperato l'identificativo del *percorso*$_G$, il *controller*$_G$ richiede a `PresentationService`, sia i nodi, sia l'oggetto rappresentante il *percorso*$_G$, chiamando i metodi `getNodes` e `getPath`.
Entrambi i metodi ritornano un oggetto di tipo `Promise` al quale il *controller*$_G$ fornisce le funzioni da invocare quando vengono risolte o rifiutate.
Se una delle due promesse viene rifiutata, il *controller*$_G$ solleva l'evento `premi-error` per segnalare all'utente che non è stato possibile caricare la presentazione.
Se invece entrambe le promesse vengono soddisfatte, il *controller*$_G$ carica nello `$scope` tutti i dati necessari alla directive `premiPresentation` per generare la presentazione.

4.2 Back-End

Nei diagrammi di sequenza viene illustrata l'interazione tra il *server*$_G$ ed i vari middleware, alcuni vengono forniti direttamente da Express mentre altri sono stati definiti dai *Progettisti*. Un middleware può terminare la propria esecuzione in tre modi diversi:

1. Termina la propria esecuzione normalmente, eseguendo la richiesta passata dal middleware precedente, e quindi termina la catena;
2. Esegue la *callback*$_G$ passatagli come parametro, in questo modo passa il controllo ai successivi middleware;
3. Se si verifica un errore allora esegue la *callback*$_G$ passando come parametro l'errore verificatosi. In questo modo la *callback*$_G$ passerà il controllo al prossimo middleware della catena che è in grado di gestire l'errore.

4.2.1 Gestione generale delle richieste

Segue un elenco ordinato dei middleware utilizzati, l'ordine di elaborazione della richiesta è determinante poiché ciascuno costituisce un handler del *design pattern*$_G$ Chain of responsibility ed ha la facoltà di interrompere la catena del chiamante:

- `express.compress()`: middleware per comprimere con il formato gzip le comunicazioni;
- `express.logger()`: middleware utilizzato per registrare un log delle richieste, utile per fare il *debugging*$_G$ dell'applicazione;
- `express.json()`: middleware che estrae dalla richiesta i parametri che sono nel formato *JSON*$_G$;
- `express.urlencoded()`: middleware che estrae dalla richiesta i parametri di tipo `www-form-encoded`, questi possono essere inviati ad esempio con una richiesta POST;
- `express.methodOverride()`: middleware utilizzato per permettere anche ai vecchi *browser*$_G$ di avere un modo per fare richieste PUT e DELETE;
- `express.cookieParser()`: middleware che analizza i cookie;
- `express.cookieSession()`: middleware per la gestione di sessioni utente basate su cookies;
- `passport.initialize()`: middleware per l'inizializzazione di Passport;
- `passport.session()`: middleware che permette di memorizzare i record della sessione utente per mantenerne lo stato di login;
- `express.route()`: permette di configurare un singolo route, nell'organizzazione interna di Premi::Back-End::App::Routers ogni modulo router contiene i routes logicamente correlati;
- `AuthenticationController`: middleware scritto dai *Progettisti* per gestire l'autenticazione. Utilizza nello specifico:
  - `passport.authenticate()`.
- `AuthorizationController`: middleware scritto dai *Progettisti* per gestire tutte quelle operazioni che richiedono un livello di autorizzazione particolare oppure che operano su informazioni da verificare prima di essere passate al prossimo middleware. In particolare, il metodo `userById`, viene utilizzato come middleware a livello di router sul parametro id (relativo all'utente) per tutte le richieste che passano questo parametro;
- `express.static()`: middleware per servire contenuti statici;

- `NotFoundHandler`: un middleware scritto dai *Progettisti* per gestire le richieste che non vengono gestite da nessun *controller*$_G$ (errore *client*$_G$ 404).
- `ErrorHandler`: middleware scritto dai *Progettisti* per gestire gli errori sollevati da altri middleware (errore *server*$_G$ 500).

Nel seguente diagramma di sequenza viene rappresentata una generica richiesta del *client*$_G$ al *server*$_G$. Le classi ed il loro comportamento sono stati progettati in funzione del *design pattern*$_G$ Chain of responsibility che viene utilizzato internamente da Express (§B.1).

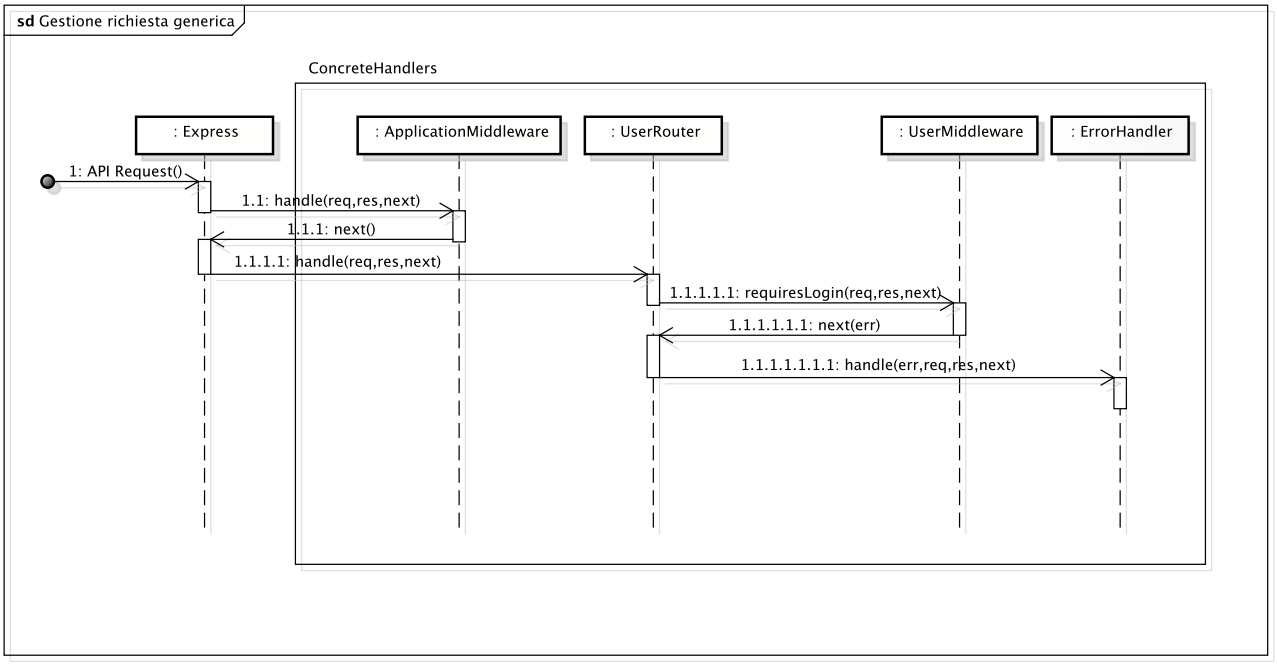


Figura 107: Gestione di una richiesta del client

4.2.2 Gestione dei fallimenti

4.2.2.1 Fallimento della procedura di registrazione

Quando un utente effettua una richiesta di registrazione e cerca di inserisce dei dati già presenti nel database viene sollevato un errore. Tale scenario rappresenta il fallimento di una richiesta di registrazione che impone, come vincolo per poter essere effettuata, che l'utente non sia autenticato e non possieda già un account nel sistema. In questo caso il modulo `AuthenticationController` invia `next(error)` per il fallimento di tale vincolo al router il quale avrà compito di reinstradarlo (indirizzandolo verso `ErrorHandler`).

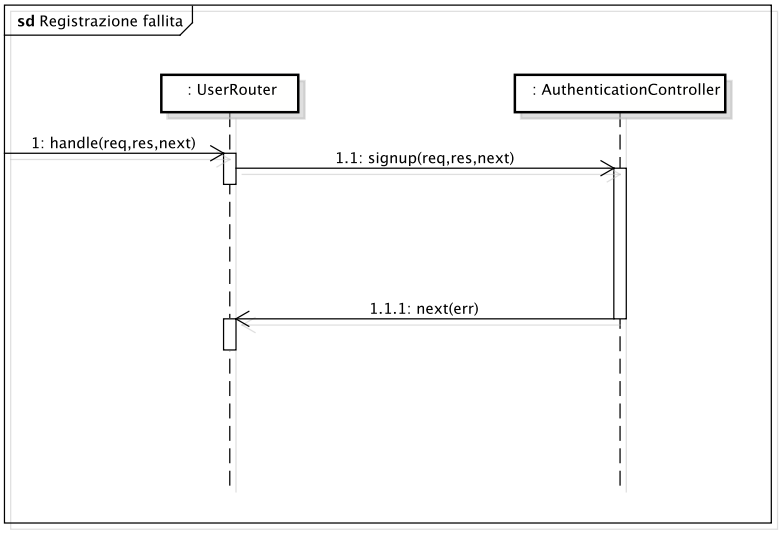


Figura 108: Fallimento della procedura di registrazione

4.2.2.2 Fallimento del vincolo utente autenticato

La maggior parte delle richieste alle risorse rese disponibili dalle API_G del $server_G$ possono essere effettuate solamente da un *utente autenticato*. Tale scenario rappresenta il fallimento del vincolo "utente autenticato". La verifica del vincolo è gestita da `AuthorizationController` che invia `next(err)` a `UserRouter` che passerà successivamente il controllo alla catena di gestione degli errori.

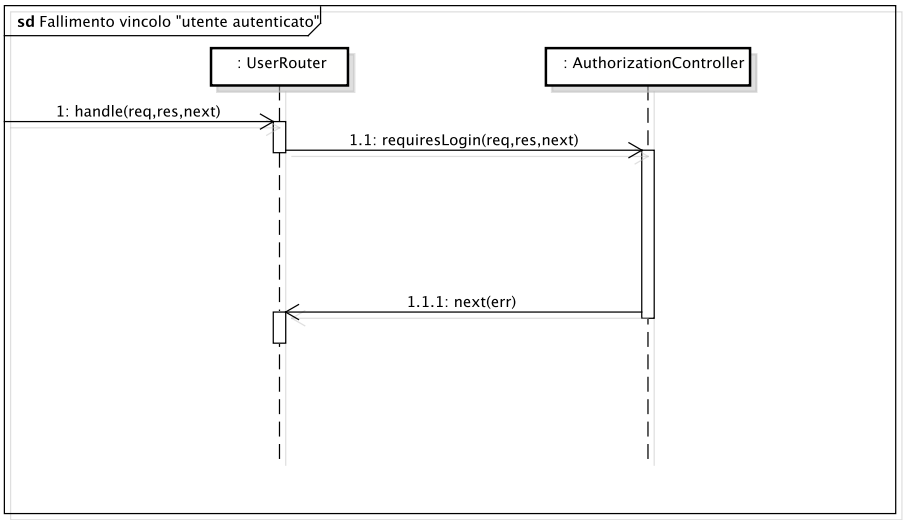


Figura 109: Fallimento del vincolo utente autenticato

4.2.2.3 Fallimento della richiesta di una lista di progetti

L'utente autenticato può richiedere una lista di $progetti_G$ presenti nel database. Nel caso in cui l'oggetto `req` sia `undefined` o malformato viene innescata la catena di gestione degli errori che

ritornerà (attraverso l'oggetto `Response`) un messaggio d'errore specifico che verrà costruito dalla classe `PremiError`. Il controllo del programma, prima di passare a questo modulo, passerà ad `ErrorHandler`. Questo perché la classe `ErrorHandler` gestisce gli errori generici e qualora vi siano degli errori specifici legati al model viene invoca `next(err)` passando quindi il controllo a `PremiError`. Da notare inoltre che la classe `ProjectController` non compare nel diagramma di sequenza nonostante sia indicata nel diagramma relativo alle relazioni tra le classi del *Back-End*$_G$. Questo perché la classe `ProjectController` serve per raggruppare tutti i controllers relativi al *progetto*$_G$ e quindi non espone alcun metodo suo (non ne ha) ma solamente i metodi dei controllers specifici del *progetto*$_G$.

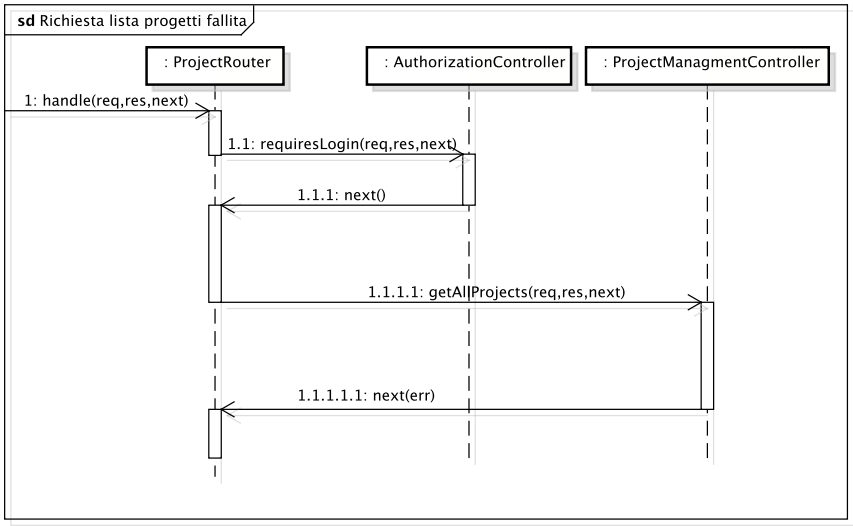


Figura 110: Fallimento della richiesta di un progetto

4.2.3 Richieste REST

4.2.3.1 DELETE /profile

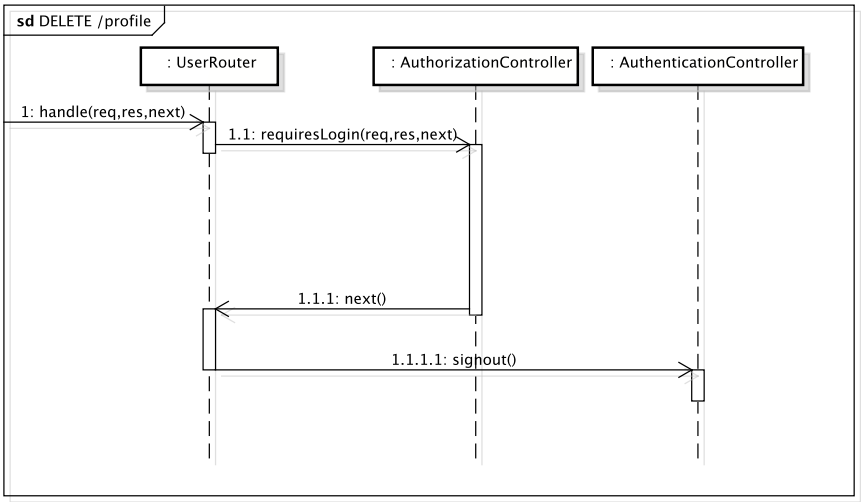


Figura 111: DELETE /profile

4.2.3.2 DELETE /projects/:projectId

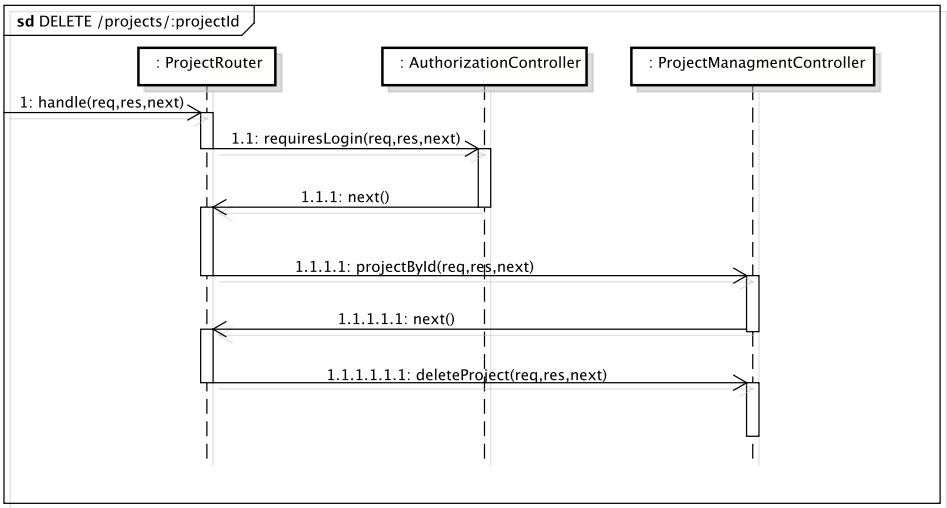


Figura 112: DELETE /projects/:projectId

4.2.3.3 PUT /projects/:projectId

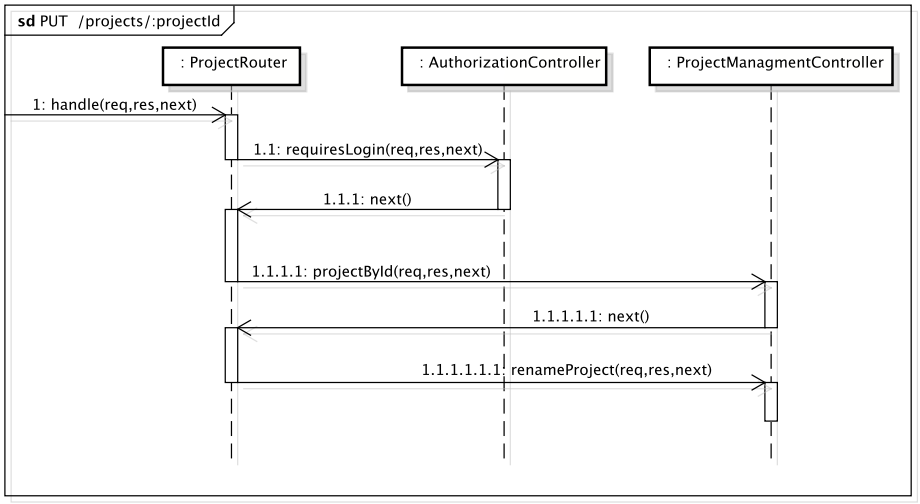


Figura 113: PUT /projects/:projectId

4.2.3.4 GET /projects/:projectId

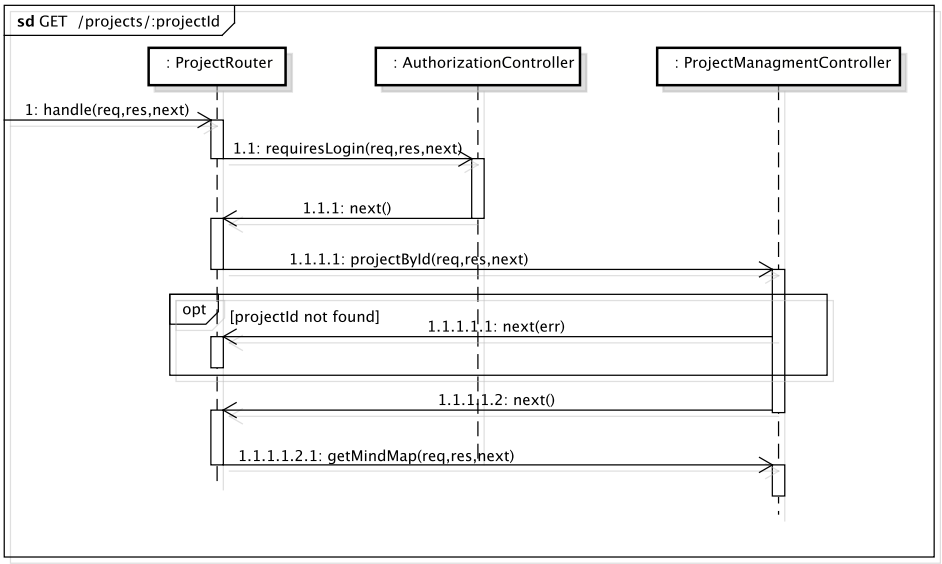


Figura 114: GET /projects/:projectId

4.2.3.5 PUT /projects/:projectId/nodes/:nodeId

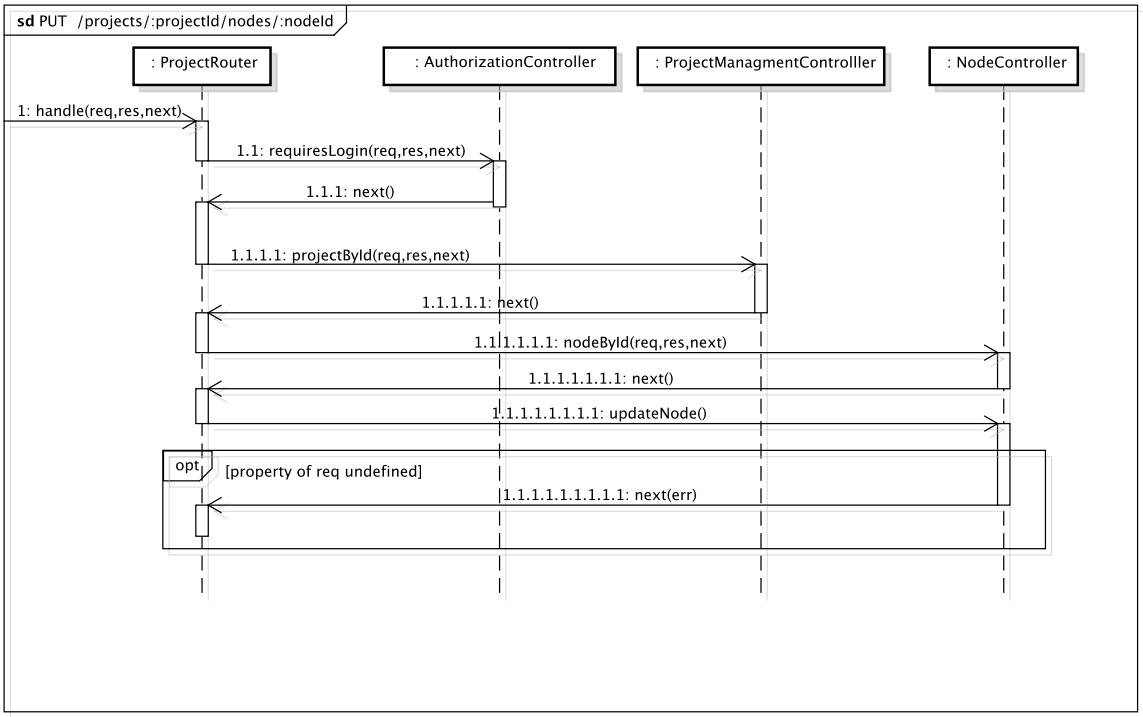


Figura 115: PUT /projects/:projectId/nodes/:nodeId

4.2.3.6 POST /projects/:projectId/associations

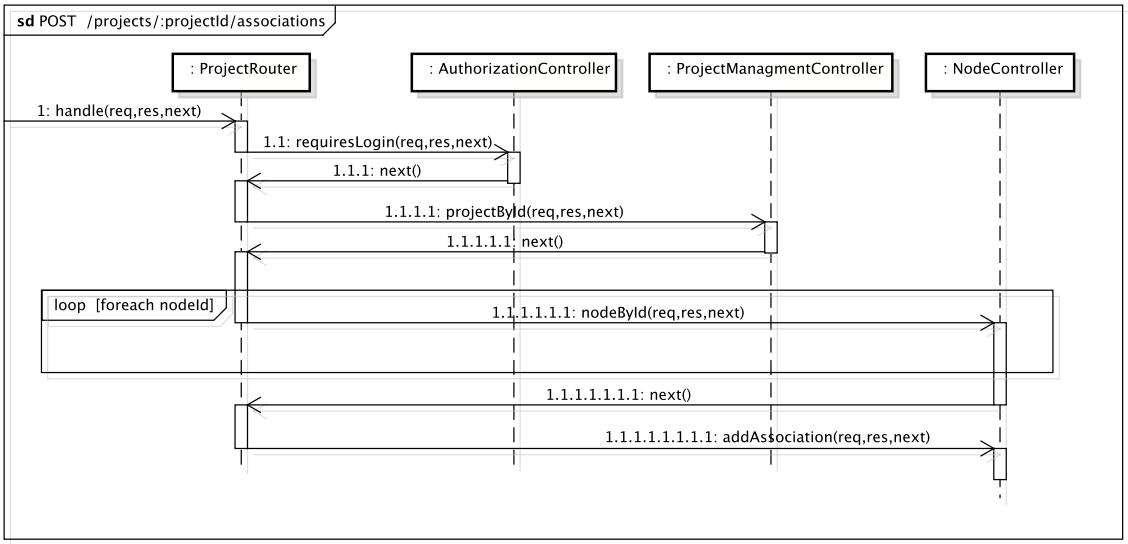


Figura 116: POST /projects/:projectId/associations

4.2.3.7 DELETE /projects/:projectId/associations/:associationId

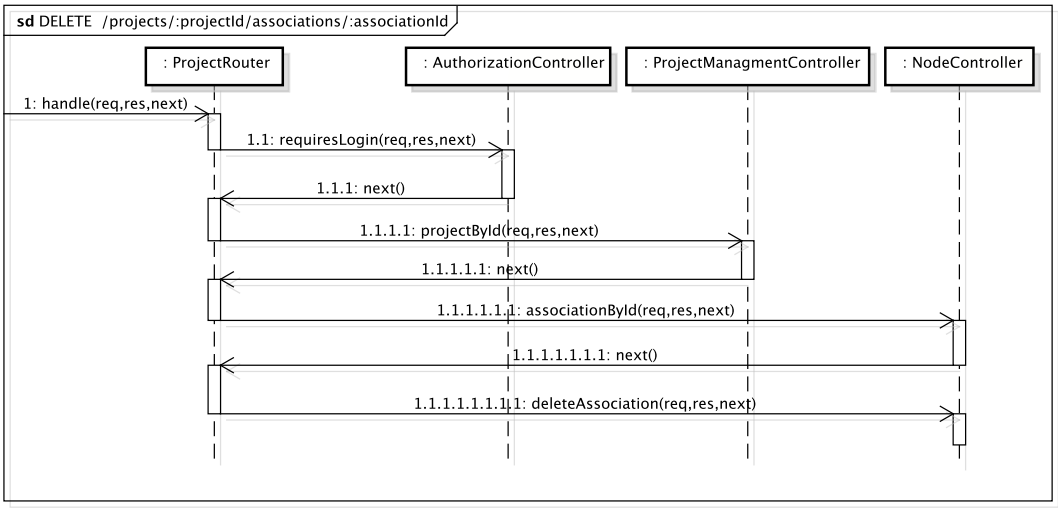


Figura 117: DELETE /projects/:projectId/associations/:associationId

4.2.3.8 PUT /projects/:projectId/path/:pathId

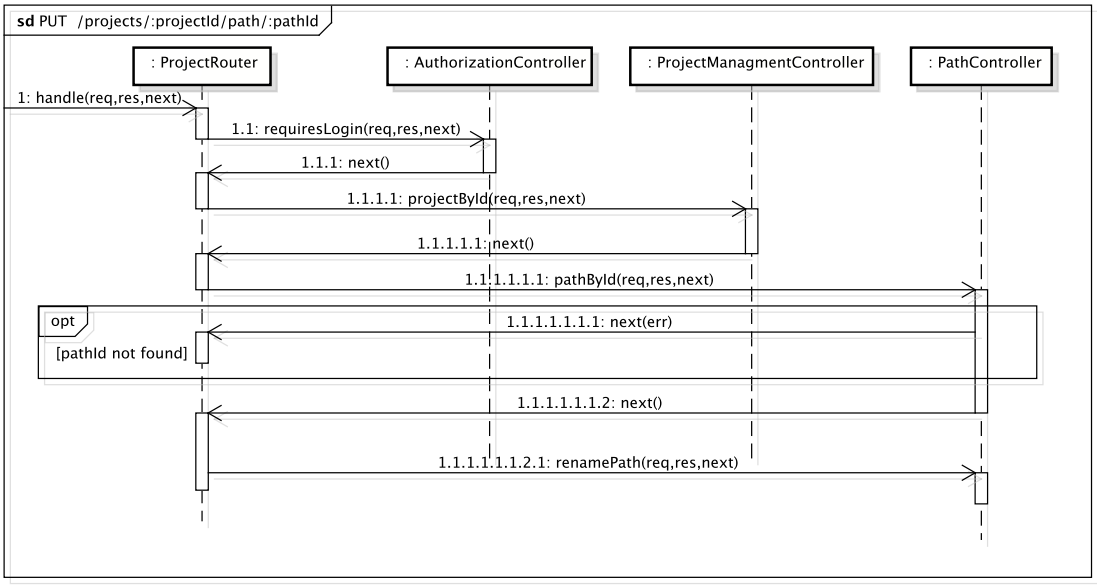


Figura 118: PUT /projects/:projectId/path/:pathId

4.2.3.9 POST /projects/:projectId/path/:pathId

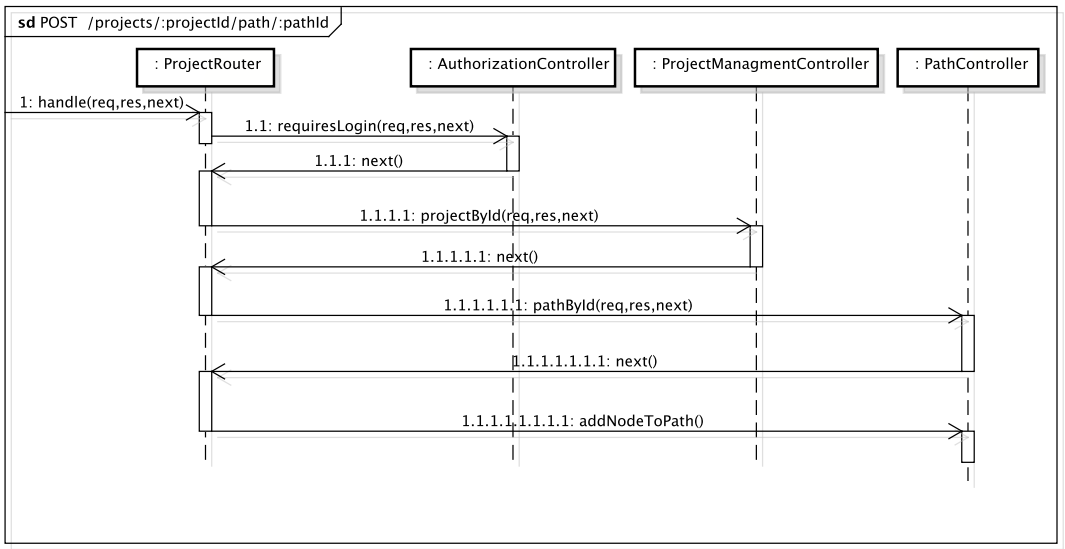


Figura 119: POST /projects/:projectId/path/:pathId

4.2.3.10 GET /projects/:projectId/premiPresentation

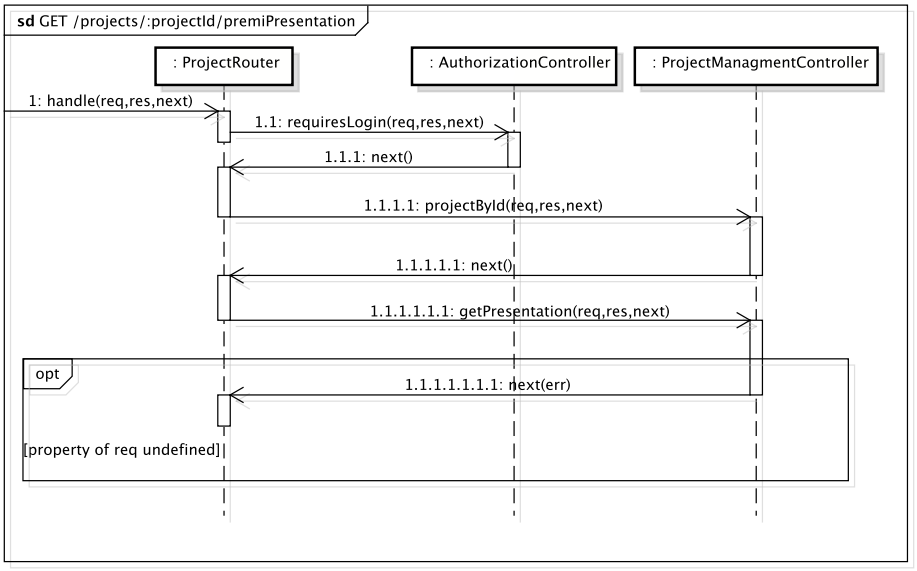


Figura 120: GET /projects/:projectId/premiPresentation

4.2.3.11 GET /static/userManual

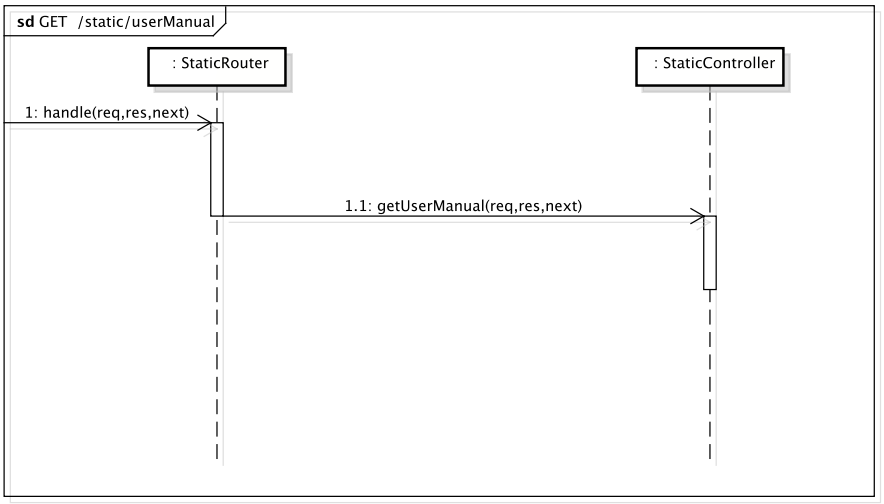


Figura 121: GET /static/userManual

5 Tracciamento

5.1 Tracciamento Classi-Requisiti

Classe	Requisiti
`Premi::Back-End::App::Controllers::-ErrorHandler`	RFO30.3 RFO30.4 RFD33 RFD35
`Premi::Back-End::App::Controllers::Errors::-PremiError`	RFO30.3 RFO30.4 RFD33 RFD35
`Premi::Back-End::App::Controllers::-NotFoundHandler`	RFD35
`Premi::Back-End::App::Controllers::-ProjectController`	RFO1 RFO1.1 RFO4 RFD4.1 RFO4.2 RFO4.2.2 RFO4.2.3 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.7 RFD4.2.3.8 RFD4.2.3.8.1 RFD4.2.3.8.2 RFO4.2.3.12 RFO4.2.3.13 RFD4.2.3.15 RFD4.2.3.16 RFO4.2.5 RFO4.2.6 RFO4.2.7 RFD4.3 RFD4.3.1 RFD4.3.2 RFD4.4 RFD4.4.1 RFD4.4.3 RFD4.4.5 RFD4.5 RFF4.6

Classe	Requisiti
	RFF4.6.1 RFF4.6.1.1 RFF4.6.1.2 RFF4.6.2 RFO7 RFD7.1 RFO10 RFO11 RFO11.2 RFO11.3 RFD22 RFD33 RFD35
`Premi::Back-End::App::Controllers::-Projects::NodeController`	RFO4 RFO4.2 RFO4.2.3 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.12 RFO4.2.3.13 RFD4.2.3.15 RFD4.2.3.16 RFO4.2.5 RFO4.2.6 RFO4.2.7 RFD22 RFD35
`Premi::Back-End::App::Controllers::-Projects::PathController`	RFO4 RFD4.3 RFD4.3.1 RFD4.3.2 RFD4.4 RFD4.4.1 RFD4.4.3 RFD4.4.5 RFD4.5 RFD22 RFD35
`Premi::Back-End::App::Controllers::-Projects::ProjectManagementController`	RFO1 RFO1.1 RFO4 RFD4.1 RFO4.2

Classe	Requisiti
	RFD4.2.3.8 RFD4.2.3.8.1 RFD4.2.3.8.2 RFF4.6 RFF4.6.1 RFF4.6.1.1 RFF4.6.1.2 RFF4.6.2 RFO10 RFO11 RFO11.2 RFO11.3 RFD22 RFD33 RFD35
`Premi::Back-End::App::Controllers::-StaticController`	RFD26 RFD35
`Premi::Back-End::App::Controllers::-UserController`	RFO30.1 RFO30.2 RFO30.3 RFO30.4 RFO32 RFD35
`Premi::Back-End::App::Controllers::Users::-AuthenticationController`	RFO30.1 RFO30.2 RFO30.3 RFO30.4 RFO32 RFD35
`Premi::Back-End::App::Controllers::Users::-AuthorizationController`	RFO30.2 RFO30.3 RFO30.4 RFO32 RFD35
`Premi::Back-End::App::Models::-NodeContentModel`	RFO4 RFO4.2.2
`Premi::Back-End::App::Models::-NodeModel`	RFO4 RFO4.2 RFO4.2.2 RFO4.2.3

Classe	Requisiti
	RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.7 RFO4.2.3.12 RFO4.2.3.13 RFD4.2.3.15 RFD4.2.3.16 RFO4.2.5 RFD22
`Premi::Back-End::App::Models::-PathModel`	RFO4 RFD4.3 RFD4.3.1 RFD4.3.2 RFD4.4 RFD4.4.1 RFD4.4.3 RFD4.4.5 RFD4.5 RFD22
`Premi::Back-End::App::Models::-ProjectModel`	RFO1 RFO1.1 RFO1.2.1 RFO1.2.1.1 RFO4 RFD4.1 RFO4.2 RFO4.2.2 RFO4.2.3 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.7 RFD4.2.3.8 RFD4.2.3.8.1 RFD4.2.3.8.2 RFO4.2.3.12 RFO4.2.3.13 RFD4.2.3.15 RFD4.2.3.16 RFO4.2.5 RFO4.2.6 RFO4.2.7 RFD4.3 RFD4.3.1 RFD4.3.2

Classe	Requisiti
	RFD4.4 RFD4.4.1 RFD4.4.3 RFD4.4.5 RFD4.5 RFF4.6 RFF4.6.1 RFF4.6.1.1 RFF4.6.1.2 RFF4.6.2 RFO7 RFD7.1 RFO10 RFO11 RFO11.2 RFO11.3 RFD22 RFD33
`Premi::Back-End::App::Models::-RelationModel`	RFO4 RFO4.2 RFO4.2.5 RFO4.2.6 RFO4.2.7 RFD22
`Premi::Back-End::App::Models::Session`	RFO30.2 RFO30.3 RFO30.4 RFO32
`Premi::Back-End::App::Models::-UserModel`	RFO30.1 RFO30.2 RFO32
`Premi::Back-End::App::Routers::-ProjectRouter`	RFO1 RFO1.1 RFO4 RFD4.1 RFO4.2 RFO4.2.2 RFO4.2.3 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.7 RFD4.2.3.8 RFD4.2.3.8.1

Classe	Requisiti
	RFD4.2.3.8.2 RFO4.2.3.12 RFO4.2.3.13 RFD4.2.3.15 RFD4.2.3.16 RFO4.2.5 RFO4.2.6 RFO4.2.7 RFD4.3.1 RFD4.3.2 RFD4.4 RFD4.4.1 RFD4.4.3 RFD4.4.5 RFD4.5 RFF4.6 RFF4.6.1 RFF4.6.1.1 RFF4.6.1.2 RFF4.6.2 RFO7 RFD7.1 RFO10 RFO11 RFO11.2 RFO11.3 RFD22 RFD33 RFD35
`Premi::Back-End::App::Routers::-StaticRouter`	RFD26 RFD35
`Premi::Back-End::App::Routers::-UserRouter`	RFO30.1 RFO30.2 RFO30.3 RFO30.4 RFO32 RFD35
`Premi::Back-End::App::Views::-UserManualView`	RFD26
`Premi::Back-End::Config::Config`	RFO36
`Premi::Back-End::Server`	RFO36
`Premi::Front-End::AppConfig`	RFO27
`Premi::Front-End::AppRouter`	RFO27
`Premi::Front-End::AppRun`	RFO27

Classe	Requisiti
	RFD35
Premi::Front-End::Controllers::-AddToPathController	RFD4.3.2 RFD4.4.1
Premi::Front-End::Controllers::-AssociationAdderController	RFO4.2 RFO4.2.6
Premi::Front-End::Controllers::-ContextMenuController	RFO4.2 RFO4.2.2 RFO4.2.5 RFO4.2.6 RFO4.2.7
Premi::Front-End::Controllers::-DashboardController	RFO1 RFO1.1 RFO1.2 RFD26 RFO27 RFO32
Premi::Front-End::Controllers::-EditableNodeContentController	RFD4.2.3.9 RFD4.2.3.10 RFF4.2.3.11 RFD4.2.3.18 RFD4.2.3.19 RFD4.2.3.20
Premi::Front-End::Controllers::-ErrorMessageController	RFO30.3 RFO30.4 RFD33
Premi::Front-End::Controllers::-HeaderController	RFF4.6 RFD15 RFF15.1 RFF15.2 RFD15.3 RFF25 RFO27 RFO32
Premi::Front-End::Controllers::-HierarchicalMenuController	RFD7.6 RFD7.6.1 RFD7.6.2
Premi::Front-End::Controllers::-LoginController	RFO30

Classe	Requisiti
	RFO30.2 RFO30.2.1 RFO30.2.2 RFO30.2.3
`Premi::Front-End::Controllers::-MindmapEditorController`	RFO4.2 RFO4.2.1 RFO5 RFD22
`Premi::Front-End::Controllers::-NodeContentsEditorController`	RFO4.2.3 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.5.1 RFO4.2.3.5.2 RFO4.2.3.7 RFO4.2.3.12 RFO4.2.3.13 RFD4.2.3.15 RFD4.2.3.16
`Premi::Front-End::Controllers::-NodeController`	RFO7.7
`Premi::Front-End::Controllers::-PathController`	RFD4.3.1 RFD4.4 RFD4.4.3 RFD4.4.5
`Premi::Front-End::Controllers::-PathsEditorController`	RFD4.4
`Premi::Front-End::Controllers::-PathsListController`	RFD4.3 RFD4.4 RFD4.5 RFD4.5.1 RFD4.7 RFD7.1
`Premi::Front-End::Controllers::-PresentationController`	RFO7 RFO7.2 RFO7.3 RFD7.6
`Premi::Front-End::Controllers::-PresentationFooterController`	RFO7 RFO7.2 RFO7.3 RFO16

Classe	Requisiti
	RFO16.1 RFF16.2 RFF16.3
`Premi::Front-End::Controllers::-PresentationViewerController`	RFO7 RFD7.1 RFO7.2 RFO7.3 RFD7.5 RFD7.6 RFD7.6.1 RFD7.6.2 RFO8 RFO9 RFD17
`Premi::Front-End::Controllers::-ProjectSettingsEditorController`	RFD4.1 RFF4.6 RFF4.6.1 RFF4.6.1.1 RFF4.6.1.2 RFF4.6.2 RFF4.6.3
`Premi::Front-End::Controllers::-ProjectsListController`	RFO4 RFD7.1 RFO10 RFO10.1 RFO10.2 RFO11 RFO11.1 RFO11.2 RFO11.3
`Premi::Front-End::Controllers::-RegistrationController`	RFO30 RFO30.1 RFO30.1.1 RFO30.1.2 RFO30.1.3
`Premi::Front-End::Controllers::-SmartMenuController`	RFD7.6 RFD7.6.1 RFD7.6.2
`Premi::Front-End::Directives::-premiAddToPath`	RFD4.3.2 RFD4.4.1

Classe	Requisiti
Premi::Front-End::Directives::-premiAssociationAdder	RFO4.2 RFO4.2.6
Premi::Front-End::Directives::-premiContextMenu	RFO4.2 RFO4.2.2 RFO4.2.5 RFO4.2.6 RFO4.2.7
Premi::Front-End::Directives::-premiEditableNodeContent	RFD4.2.3.9 RFD4.2.3.10 RFF4.2.3.11 RFD4.2.3.18 RFD4.2.3.19 RFD4.2.3.20
Premi::Front-End::Directives::-premiErrorMessage	RFO30.3 RFO30.4 RFD33
Premi::Front-End::Directives::-premiHeader	RFF4.6 RFD15 RFF15.1 RFF15.2 RFD15.3 RFF25 RFO27
Premi::Front-End::Directives::-premiHierarchicalMenu	RFD7.6 RFD7.6.1 RFD7.6.2
Premi::Front-End::Directives::-premiMindmap	RFD22
Premi::Front-End::Directives::-premiNode	RFO7.7
Premi::Front-End::Directives::-premiNodeContent	RFO7.7
Premi::Front-End::Directives::-premiNodeContentsEditor	RFO4.2.3 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.5.1 RFO4.2.3.5.2 RFO4.2.3.7 RFO4.2.3.12

Classe	Requisiti
	RFO4.2.3.13 RFD4.2.3.15 RFD4.2.3.16
`Premi::Front-End::Directives::-premiPath`	RFD4.3.1 RFD4.4 RFD4.4.3 RFD4.4.5
`Premi::Front-End::Directives::-premiPathsList`	RFD4.3 RFD4.3.3 RFD4.5 RFD4.5.1 RFD4.7 RFD7.1
`Premi::Front-End::Directives::-premiPresentation`	RFO7 RFO7.2 RFO7.3 RFD7.6 RFD7.6.1 RFD7.6.2
`Premi::Front-End::Directives::-premiPresentationFooter`	RFO7 RFO7.2 RFO7.3 RFO16 RFO16.1 RFF16.2 RFF16.3
`Premi::Front-End::Directives::-premiProjectSettingsEditor`	RFD4.1 RFF4.6 RFF4.6.1 RFF4.6.1.1 RFF4.6.1.2 RFF4.6.2 RFF4.6.3
`Premi::Front-End::Directives::-premiProjectsList`	RFO4 RFD7.1 RFO10 RFO10.1 RFO10.2 RFO11 RFO11.1

Classe	Requisiti
	RFO11.2 RFO11.3
`Premi::Front-End::Directives::-premiSmartMenu`	RFD7.6 RFD7.6.1 RFD7.6.2
`Premi::Front-End::Model::ErrorInfo`	RFO30.3 RFO30.4 RFD33 RFD35
`Premi::Front-End::Model::Node`	RFO4 RFO4.2 RFO4.2.3 RFO4.2.3.1 RFO4.2.3.2 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.5.1 RFO4.2.3.5.2 RFO4.2.3.7 RFD4.2.3.15 RFD4.2.3.16 RFD4.2.3.17 RFD4.2.3.18 RFD4.2.3.19 RFD4.2.3.20 RFO5 RFD6 RFO7.7 RFD22
`Premi::Front-End::Model::NodeContent`	RFO4 RFO4.2 RFO4.2.3.1 RFO4.2.3.2 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.5.1 RFO4.2.3.5.2 RFO4.2.3.7 RFD4.2.3.15 RFD4.2.3.16 RFD4.2.3.17 RFD4.2.3.18 RFD4.2.3.19 RFD4.2.3.20 RFO5

Classe	Requisiti
	RFD6 RFO7.7 RFD22
`Premi::Front-End::Model::NodeReference`	RFO4 RFD4.4 RFD4.4.1 RFD4.4.3 RFO5 RFD6
`Premi::Front-End::Model::Path`	RFO4 RFD4.4 RFD4.4.1 RFD4.4.3 RFD4.4.5 RFF4.6.2 RFD4.7 RFO5 RFD6
`Premi::Front-End::Model::-PresentationNode`	RFO7 RFO7.2 RFO7.3 RFO7.7
`Premi::Front-End::Model::Project`	RFO4 RFD4.1 RFF4.6 RFF4.6.1 RFF4.6.1.1 RFF4.6.1.2 RFF4.6.2 RFO5 RFD6
`Premi::Front-End::Services::-AuthenticationService`	RFO30 RFO30.1 RFO30.1.1 RFO30.1.2 RFO30.1.3 RFO30.2 RFO30.2.1 RFO30.2.2 RFO30.2.3 RFO32
`Premi::Front-End::Services::-MindmapAdapterService`	RFO1 RFO1.2

Classe	Requisiti
	RFO4 RFO4.2 RFO4.2.1 RFO4.2.2 RFO4.2.3 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.5.1 RFO4.2.3.5.2 RFO4.2.3.7 RFD4.2.3.15 RFD4.2.3.16 RFD4.2.3.18 RFD4.2.3.19 RFD4.2.3.20 RFO4.2.5 RFO4.2.6 RFO4.2.7 RFO4.2.8 RFO5 RFD6 RFO7.7 RFD22
`Premi::Front-End::Services::-MindmapService`	RFO4 RFO4.2 RFO4.2.1 RFO4.2.2 RFO4.2.3 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.5.2 RFO4.2.3.7 RFD4.2.3.15 RFD4.2.3.16 RFD4.2.3.18 RFD4.2.3.19 RFD4.2.3.20 RFO4.2.5 RFO4.2.6 RFO4.2.7 RFO4.2.8 RFO5 RFD6 RFO7.7 RFD22

Classe	Requisiti
	RFF25.1
`Premi::Front-End::Services::-PathService`	RFO4 RFD4.3 RFD4.3.1 RFD4.3.3 RFD4.4 RFD4.4.1 RFD4.4.3 RFD4.4.5 RFD4.5 RFD4.5.1 RFD4.7 RFO5 RFD6 RFD23 RFF25.1
`Premi::Front-End::Services::-PresentationService`	RFD4.7 RFO7 RFD7.1 RFD7.4.1 RFD7.4.2 RFD7.6.2 RFO7.7 RFO8 RFO9 RFD16.4
`Premi::Front-End::Services::-ProjectService`	RFO1 RFO1.1 RFO1.2 RFO2 RFD3 RFO4 RFD4.1 RFF4.6 RFF4.6.3 RFO5 RFD6 RFO10 RFO10.1 RFO10.2 RFO11 RFO11.1 RFO11.2

Classe	Requisiti
	RFO11.3 RFF25.1
`Premi::Front-End::Views::DashboardView`	RFO1 RFO1.1 RFO1.2 RFD26 RFO27 RFO32
`Premi::Front-End::Views::LoginView`	RFO30 RFO30.2 RFO30.2.1 RFO30.2.2 RFO30.2.3
`Premi::Front-End::Views::-MindmapEditorView`	RFO4.2 RFO4.2.1 RFO5 RFD22
`Premi::Front-End::Views::-PathsEditorView`	RFD4.4
`Premi::Front-End::Views::-PresentationView`	RFO7 RFD7.1 RFO7.2 RFO7.3 RFD7.5 RFD7.6 RFD7.6.1 RFD7.6.2 RFO8 RFO9 RFD14 RFD14.1 RFD14.2 RFD14.3 RFO16 RFO16.1 RFF16.2 RFF16.3 RFD17 RFO18
`Premi::Front-End::Views::-RegistrationView`	RFO30 RFO30.1 RFO30.1.1 RFO30.1.2

Classe	Requisiti
	RFO30.1.3

Tabella 2: Tracciamento Classi-Requisiti

5.2 Tracciamento Requisiti-Classi

Requisito	Classi
RFO1	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-DashboardController Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-ProjectService Premi::Front-End::Views::DashboardView
RFO1.1	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-DashboardController Premi::Front-End::Services::-ProjectService Premi::Front-End::Views::DashboardView
RFO1.2	Premi::Front-End::Controllers::-DashboardController Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-ProjectService Premi::Front-End::Views::DashboardView
RFO1.2.1	Premi::Back-End::App::Models::-ProjectModel
RFO1.2.1.1	Premi::Back-End::App::Models::-ProjectModel
RFO2	Premi::Front-End::Services::-ProjectService
RFD3	Premi::Front-End::Services::-ProjectService

Requisito	Classi
RFO4	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::NodeController Premi::Back-End::App::Controllers::-Projects::PathController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-NodeContentModel Premi::Back-End::App::Models::-NodeModel Premi::Back-End::App::Models::-PathModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Models::-RelationModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-ProjectsListController Premi::Front-End::Directives::-premiProjectsList Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent Premi::Front-End::Model::NodeReference Premi::Front-End::Model::Path Premi::Front-End::Model::Project Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService Premi::Front-End::Services::-PathService Premi::Front-End::Services::-ProjectService
RFD4.1	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter

Requisito	Classi
	Premi::Front-End::Controllers::-ProjectSettingsEditorController Premi::Front-End::Directives::-premiProjectSettingsEditor Premi::Front-End::Model::Project Premi::Front-End::Services::-ProjectService
RFO4.2	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::NodeController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-NodeModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Models::-RelationModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-AssociationAdderController Premi::Front-End::Controllers::-ContextMenuController Premi::Front-End::Controllers::-MindmapEditorController Premi::Front-End::Directives::-premiAssociationAdder Premi::Front-End::Directives::-premiContextMenu Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService Premi::Front-End::Views::-MindmapEditorView
RFO4.2.1	Premi::Front-End::Controllers::-MindmapEditorController Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService

Requisito	Classi
	Premi::Front-End::Views::-MindmapEditorView
RFO4.2.2	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Models::-NodeContentModel Premi::Back-End::App::Models::-NodeModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-ContextMenuController Premi::Front-End::Directives::-premiContextMenu Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService
RFO4.2.3	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::NodeController Premi::Back-End::App::Models::-NodeModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-NodeContentsEditorController Premi::Front-End::Directives::-premiNodeContentsEditor Premi::Front-End::Model::Node Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService
RFO4.2.3.1	Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent
RFO4.2.3.2	Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent
RFO4.2.3.4	Premi::Back-End::App::Controllers::-ProjectController

Requisito	Classi
	Premi::Back-End::App::Controllers::-Projects::NodeController Premi::Back-End::App::Models::-NodeModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-NodeContentsEditorController Premi::Front-End::Directives::-premiNodeContentsEditor Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService
RFO4.2.3.5	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::NodeController Premi::Back-End::App::Models::-NodeModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-NodeContentsEditorController Premi::Front-End::Directives::-premiNodeContentsEditor Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService
RFO4.2.3.5.1	Premi::Front-End::Controllers::-NodeContentsEditorController Premi::Front-End::Directives::-premiNodeContentsEditor Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent Premi::Front-End::Services::-MindmapAdapterService

Requisito	Classi
RFO4.2.3.5.2	Premi::Front-End::Controllers::-NodeContentsEditorController Premi::Front-End::Directives::-premiNodeContentsEditor Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService
RFO4.2.3.7	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Models::-NodeModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-NodeContentsEditorController Premi::Front-End::Directives::-premiNodeContentsEditor Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService
RFD4.2.3.8	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter
RFD4.2.3.8.1	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter
RFD4.2.3.8.2	Premi::Back-End::App::Controllers::-ProjectController

Requisito	Classi
	Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter
RFD4.2.3.9	Premi::Front-End::Controllers::-EditableNodeContentController Premi::Front-End::Directives::-premiEditableNodeContent
RFD4.2.3.10	Premi::Front-End::Controllers::-EditableNodeContentController Premi::Front-End::Directives::-premiEditableNodeContent
RFF4.2.3.11	Premi::Front-End::Controllers::-EditableNodeContentController Premi::Front-End::Directives::-premiEditableNodeContent
RFO4.2.3.12	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::NodeController Premi::Back-End::App::Models::-NodeModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-NodeContentsEditorController Premi::Front-End::Directives::-premiNodeContentsEditor
RFO4.2.3.13	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::NodeController Premi::Back-End::App::Models::-NodeModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-NodeContentsEditorController Premi::Front-End::Directives::-premiNodeContentsEditor

Requisito	Classi
RFD4.2.3.15	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::NodeController Premi::Back-End::App::Models::-NodeModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-NodeContentsEditorController Premi::Front-End::Directives::-premiNodeContentsEditor Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService
RFD4.2.3.16	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::NodeController Premi::Back-End::App::Models::-NodeModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-NodeContentsEditorController Premi::Front-End::Directives::-premiNodeContentsEditor Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService
RFD4.2.3.17	Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent
RFD4.2.3.18	Premi::Front-End::Controllers::-EditableNodeContentController Premi::Front-End::Directives::-premiEditableNodeContent

Requisito	Classi
	Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent Premi::Front-End::Services::- MindmapAdapterService Premi::Front-End::Services::- MindmapService
RFD4.2.3.19	Premi::Front-End::Controllers::- EditableNodeContentController Premi::Front-End::Directives::- premiEditableNodeContent Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent Premi::Front-End::Services::- MindmapAdapterService Premi::Front-End::Services::- MindmapService
RFD4.2.3.20	Premi::Front-End::Controllers::- EditableNodeContentController Premi::Front-End::Directives::- premiEditableNodeContent Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent Premi::Front-End::Services::- MindmapAdapterService Premi::Front-End::Services::- MindmapService
RFO4.2.5	Premi::Back-End::App::Controllers::- ProjectController Premi::Back-End::App::Controllers::- Projects::NodeController Premi::Back-End::App::Models::- NodeModel Premi::Back-End::App::Models::- ProjectModel Premi::Back-End::App::Models::- RelationModel Premi::Back-End::App::Routers::- ProjectRouter Premi::Front-End::Controllers::- ContextMenuController Premi::Front-End::Directives::- premiContextMenu Premi::Front-End::Services::- MindmapAdapterService Premi::Front-End::Services::- MindmapService

Requisito	Classi
RFO4.2.6	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::NodeController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Models::-RelationModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-AssociationAdderController Premi::Front-End::Controllers::-ContextMenuController Premi::Front-End::Directives::-premiAssociationAdder Premi::Front-End::Directives::-premiContextMenu Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService
RFO4.2.7	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::NodeController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Models::-RelationModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-ContextMenuController Premi::Front-End::Directives::-premiContextMenu Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService
RFO4.2.8	Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService
RFD4.3	Premi::Back-End::App::Controllers::-ProjectController

Requisito	Classi
	Premi::Back-End::App::Controllers::-Projects::PathController Premi::Back-End::App::Models::-PathModel Premi::Back-End::App::Models::-ProjectModel Premi::Front-End::Controllers::-PathsListController Premi::Front-End::Directives::-premiPathsList Premi::Front-End::Services::-PathService
RFD4.3.1	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::PathController Premi::Back-End::App::Models::-PathModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-PathController Premi::Front-End::Directives::-premiPath Premi::Front-End::Services::-PathService
RFD4.3.2	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::PathController Premi::Back-End::App::Models::-PathModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-AddToPathController Premi::Front-End::Directives::-premiAddToPath
RFD4.3.3	Premi::Front-End::Directives::-premiPathsList Premi::Front-End::Services::-PathService

Requisito	Classi
RFD4.4	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::PathController Premi::Back-End::App::Models::-PathModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-PathController Premi::Front-End::Controllers::-PathsEditorController Premi::Front-End::Controllers::-PathsListController Premi::Front-End::Directives::-premiPath Premi::Front-End::Model::NodeReference Premi::Front-End::Model::Path Premi::Front-End::Services::-PathService Premi::Front-End::Views::-PathsEditorView
RFD4.4.1	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::PathController Premi::Back-End::App::Models::-PathModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-AddToPathController Premi::Front-End::Directives::-premiAddToPath Premi::Front-End::Model::NodeReference Premi::Front-End::Model::Path Premi::Front-End::Services::-PathService
RFD4.4.3	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::PathController

Requisito	Classi
	Premi::Back-End::App::Models::-PathModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-PathController Premi::Front-End::Directives::-premiPath Premi::Front-End::Model::NodeReference Premi::Front-End::Model::Path Premi::Front-End::Services::-PathService
RFD4.4.5	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::PathController Premi::Back-End::App::Models::-PathModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-PathController Premi::Front-End::Directives::-premiPath Premi::Front-End::Model::Path Premi::Front-End::Services::-PathService
RFD4.5	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::PathController Premi::Back-End::App::Models::-PathModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-PathsListController Premi::Front-End::Directives::-premiPathsList

Requisito	Classi
	Premi::Front-End::Services::-PathService
RFD4.5.1	Premi::Front-End::Controllers::-PathsListController Premi::Front-End::Directives::-premiPathsList Premi::Front-End::Services::-PathService
RFF4.6	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-HeaderController Premi::Front-End::Controllers::-ProjectSettingsEditorController Premi::Front-End::Directives::-premiHeader Premi::Front-End::Directives::-premiProjectSettingsEditor Premi::Front-End::Model::Project Premi::Front-End::Services::-ProjectService
RFF4.6.1	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-ProjectSettingsEditorController Premi::Front-End::Directives::-premiProjectSettingsEditor Premi::Front-End::Model::Project
RFF4.6.1.1	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-ProjectModel

Requisito	Classi
	Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-ProjectSettingsEditorController Premi::Front-End::Directives::-premiProjectSettingsEditor Premi::Front-End::Model::Project
RFF4.6.1.2	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-ProjectSettingsEditorController Premi::Front-End::Directives::-premiProjectSettingsEditor Premi::Front-End::Model::Project
RFF4.6.2	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-ProjectSettingsEditorController Premi::Front-End::Directives::-premiProjectSettingsEditor Premi::Front-End::Model::Path Premi::Front-End::Model::Project
RFF4.6.3	Premi::Front-End::Controllers::-ProjectSettingsEditorController Premi::Front-End::Directives::-premiProjectSettingsEditor Premi::Front-End::Services::-ProjectService
RFD4.7	Premi::Front-End::Controllers::-PathsListController Premi::Front-End::Directives::-premiPathsList Premi::Front-End::Model::Path

Requisito	Classi
	Premi::Front-End::Services::-PathService Premi::Front-End::Services::-PresentationService
RFO5	Premi::Front-End::Controllers::-MindmapEditorController Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent Premi::Front-End::Model::NodeReference Premi::Front-End::Model::Path Premi::Front-End::Model::Project Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService Premi::Front-End::Services::-PathService Premi::Front-End::Services::-ProjectService Premi::Front-End::Views::-MindmapEditorView
RFD6	Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent Premi::Front-End::Model::NodeReference Premi::Front-End::Model::Path Premi::Front-End::Model::Project Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService Premi::Front-End::Services::-PathService Premi::Front-End::Services::-ProjectService
RFO7	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-PresentationController Premi::Front-End::Controllers::-PresentationFooterController Premi::Front-End::Controllers::-PresentationViewerController

Requisito	Classi
	Premi::Front-End::Directives::-premiPresentation Premi::Front-End::Directives::-premiPresentationFooter Premi::Front-End::Model::-PresentationNode Premi::Front-End::Services::-PresentationService Premi::Front-End::Views::-PresentationView
RFD7.1	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-PathsListController Premi::Front-End::Controllers::-PresentationViewerController Premi::Front-End::Controllers::-ProjectsListController Premi::Front-End::Directives::-premiPathsList Premi::Front-End::Directives::-premiProjectsList Premi::Front-End::Services::-PresentationService Premi::Front-End::Views::-PresentationView
RFO7.2	Premi::Front-End::Controllers::-PresentationController Premi::Front-End::Controllers::-PresentationFooterController Premi::Front-End::Controllers::-PresentationViewerController Premi::Front-End::Directives::-premiPresentation Premi::Front-End::Directives::-premiPresentationFooter Premi::Front-End::Model::-PresentationNode Premi::Front-End::Views::-PresentationView
RFO7.3	Premi::Front-End::Controllers::-PresentationController

Requisito	Classi
	Premi::Front-End::Controllers::-PresentationFooterController Premi::Front-End::Controllers::-PresentationViewerController Premi::Front-End::Directives::-premiPresentation Premi::Front-End::Directives::-premiPresentationFooter Premi::Front-End::Model::-PresentationNode Premi::Front-End::Views::-PresentationView
RFD7.4.1	Premi::Front-End::Services::-PresentationService
RFD7.4.2	Premi::Front-End::Services::-PresentationService
RFD7.5	Premi::Front-End::Controllers::-PresentationViewerController Premi::Front-End::Views::-PresentationView
RFD7.6	Premi::Front-End::Controllers::-HierarchicalMenuController Premi::Front-End::Controllers::-PresentationController Premi::Front-End::Controllers::-PresentationViewerController Premi::Front-End::Controllers::-SmartMenuController Premi::Front-End::Directives::-premiHierarchicalMenu Premi::Front-End::Directives::-premiPresentation Premi::Front-End::Directives::-premiSmartMenu Premi::Front-End::Views::-PresentationView
RFD7.6.1	Premi::Front-End::Controllers::-HierarchicalMenuController Premi::Front-End::Controllers::-PresentationViewerController Premi::Front-End::Controllers::-SmartMenuController Premi::Front-End::Directives::-premiHierarchicalMenu Premi::Front-End::Directives::-premiPresentation

Requisito	Classi
	Premi::Front-End::Directives::-premiSmartMenu Premi::Front-End::Views::-PresentationView
RFD7.6.2	Premi::Front-End::Controllers::-HierarchicalMenuController Premi::Front-End::Controllers::-PresentationViewerController Premi::Front-End::Controllers::-SmartMenuController Premi::Front-End::Directives::-premiHierarchicalMenu Premi::Front-End::Directives::-premiPresentation Premi::Front-End::Directives::-premiSmartMenu Premi::Front-End::Services::-PresentationService Premi::Front-End::Views::-PresentationView
RFO7.7	Premi::Front-End::Controllers::-NodeController Premi::Front-End::Directives::-premiNode Premi::Front-End::Directives::-premiNodeContent Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent Premi::Front-End::Model::-PresentationNode Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService Premi::Front-End::Services::-PresentationService
RFO8	Premi::Front-End::Controllers::-PresentationViewerController Premi::Front-End::Services::-PresentationService Premi::Front-End::Views::-PresentationView
RFO9	Premi::Front-End::Controllers::-PresentationViewerController Premi::Front-End::Services::-PresentationService

Requisito	Classi
	Premi::Front-End::Views::-PresentationView
RFO10	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-ProjectsListController Premi::Front-End::Directives::-premiProjectsList Premi::Front-End::Services::-ProjectService
RFO10.1	Premi::Front-End::Controllers::-ProjectsListController Premi::Front-End::Directives::-premiProjectsList Premi::Front-End::Services::-ProjectService
RFO10.2	Premi::Front-End::Controllers::-ProjectsListController Premi::Front-End::Directives::-premiProjectsList Premi::Front-End::Services::-ProjectService
RFO11	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-ProjectsListController Premi::Front-End::Directives::-premiProjectsList Premi::Front-End::Services::-ProjectService
RFO11.1	Premi::Front-End::Controllers::-ProjectsListController Premi::Front-End::Directives::-premiProjectsList

Requisito	Classi
	Premi::Front-End::Services::-ProjectService
RFO11.2	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-ProjectsListController Premi::Front-End::Directives::-premiProjectsList Premi::Front-End::Services::-ProjectService
RFO11.3	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-ProjectsListController Premi::Front-End::Directives::-premiProjectsList Premi::Front-End::Services::-ProjectService
RFD14	Premi::Front-End::Views::-PresentationView
RFD14.1	Premi::Front-End::Views::-PresentationView
RFD14.2	Premi::Front-End::Views::-PresentationView
RFD14.3	Premi::Front-End::Views::-PresentationView
RFD15	Premi::Front-End::Controllers::-HeaderController Premi::Front-End::Directives::-premiHeader
RFF15.1	Premi::Front-End::Controllers::-HeaderController Premi::Front-End::Directives::-premiHeader

Requisito	Classi
RFF15.2	Premi::Front-End::Controllers::-HeaderController Premi::Front-End::Directives::-premiHeader
RFD15.3	Premi::Front-End::Controllers::-HeaderController Premi::Front-End::Directives::-premiHeader
RFO16	Premi::Front-End::Controllers::-PresentationFooterController Premi::Front-End::Directives::-premiPresentationFooter Premi::Front-End::Views::-PresentationView
RFO16.1	Premi::Front-End::Controllers::-PresentationFooterController Premi::Front-End::Directives::-premiPresentationFooter Premi::Front-End::Views::-PresentationView
RFF16.2	Premi::Front-End::Controllers::-PresentationFooterController Premi::Front-End::Directives::-premiPresentationFooter Premi::Front-End::Views::-PresentationView
RFF16.3	Premi::Front-End::Controllers::-PresentationFooterController Premi::Front-End::Directives::-premiPresentationFooter Premi::Front-End::Views::-PresentationView
RFD16.4	Premi::Front-End::Services::-PresentationService
RFD17	Premi::Front-End::Controllers::-PresentationViewerController Premi::Front-End::Views::-PresentationView
RFO18	Premi::Front-End::Views::-PresentationView
RFD22	Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::NodeController Premi::Back-End::App::Controllers::-Projects::PathController

Requisito	Classi
	Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-NodeModel Premi::Back-End::App::Models::-PathModel Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Models::-RelationModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-MindmapEditorController Premi::Front-End::Directives::-premiMindmap Premi::Front-End::Model::Node Premi::Front-End::Model::NodeContent Premi::Front-End::Services::-MindmapAdapterService Premi::Front-End::Services::-MindmapService Premi::Front-End::Views::-MindmapEditorView
RFD23	Premi::Front-End::Services::-PathService
RFF25	Premi::Front-End::Controllers::-HeaderController Premi::Front-End::Directives::-premiHeader
RFF25.1	Premi::Front-End::Services::-MindmapService Premi::Front-End::Services::-PathService Premi::Front-End::Services::-ProjectService
RFD26	Premi::Back-End::App::Controllers::-StaticController Premi::Back-End::App::Routers::-StaticRouter Premi::Back-End::App::Views::-UserManualView Premi::Front-End::Controllers::-DashboardController Premi::Front-End::Views::DashboardView
RFO27	Premi::Front-End::AppConfig

Requisito	Classi
	Premi::Front-End::AppRouter Premi::Front-End::AppRun Premi::Front-End::Controllers::-DashboardController Premi::Front-End::Controllers::-HeaderController Premi::Front-End::Directives::-premiHeader Premi::Front-End::Views::DashboardView
RFO30	Premi::Front-End::Controllers::-LoginController Premi::Front-End::Controllers::-RegistrationController Premi::Front-End::Services::-AuthenticationService Premi::Front-End::Views::LoginView Premi::Front-End::Views::-RegistrationView
RFO30.1	Premi::Back-End::App::Controllers::-UserController Premi::Back-End::App::Controllers::Users::-AuthenticationController Premi::Back-End::App::Models::-UserModel Premi::Back-End::App::Routers::-UserRouter Premi::Front-End::Controllers::-RegistrationController Premi::Front-End::Services::-AuthenticationService Premi::Front-End::Views::-RegistrationView
RFO30.1.1	Premi::Front-End::Controllers::-RegistrationController Premi::Front-End::Services::-AuthenticationService Premi::Front-End::Views::-RegistrationView
RFO30.1.2	Premi::Front-End::Controllers::-RegistrationController Premi::Front-End::Services::-AuthenticationService Premi::Front-End::Views::-RegistrationView
RFO30.1.3	Premi::Front-End::Controllers::-RegistrationController

Requisito	Classi
	Premi::Front-End::Services::-AuthenticationService Premi::Front-End::Views::-RegistrationView
RFO30.2	Premi::Back-End::App::Controllers::-UserController Premi::Back-End::App::Controllers::Users::-AuthenticationController Premi::Back-End::App::Controllers::Users::-AuthorizationController Premi::Back-End::App::Models::Session Premi::Back-End::App::Models::-UserModel Premi::Back-End::App::Routers::-UserRouter Premi::Front-End::Controllers::-LoginController Premi::Front-End::Services::-AuthenticationService Premi::Front-End::Views::LoginView
RFO30.2.1	Premi::Front-End::Controllers::-LoginController Premi::Front-End::Services::-AuthenticationService Premi::Front-End::Views::LoginView
RFO30.2.2	Premi::Front-End::Controllers::-LoginController Premi::Front-End::Services::-AuthenticationService Premi::Front-End::Views::LoginView
RFO30.2.3	Premi::Front-End::Controllers::-LoginController Premi::Front-End::Services::-AuthenticationService Premi::Front-End::Views::LoginView
RFO30.3	Premi::Back-End::App::Controllers::-ErrorHandler Premi::Back-End::App::Controllers::Errors::-PremiError Premi::Back-End::App::Controllers::-UserController Premi::Back-End::App::Controllers::Users::-AuthenticationController Premi::Back-End::App::Controllers::Users::-AuthorizationController Premi::Back-End::App::Models::Session

Requisito	Classi
	Premi::Back-End::App::Routers::-UserRouter Premi::Front-End::Controllers::-ErrorMessageController Premi::Front-End::Directives::-premiErrorMessage Premi::Front-End::Model::ErrorInfo
RFO30.4	Premi::Back-End::App::Controllers::-ErrorHandler Premi::Back-End::App::Controllers::Errors::-PremiError Premi::Back-End::App::Controllers::-UserController Premi::Back-End::App::Controllers::Users::-AuthenticationController Premi::Back-End::App::Controllers::Users::-AuthorizationController Premi::Back-End::App::Models::Session Premi::Back-End::App::Routers::-UserRouter Premi::Front-End::Controllers::-ErrorMessageController Premi::Front-End::Directives::-premiErrorMessage Premi::Front-End::Model::ErrorInfo
RFO32	Premi::Back-End::App::Controllers::-UserController Premi::Back-End::App::Controllers::Users::-AuthenticationController Premi::Back-End::App::Controllers::Users::-AuthorizationController Premi::Back-End::App::Models::Session Premi::Back-End::App::Models::-UserModel Premi::Back-End::App::Routers::-UserRouter Premi::Front-End::Controllers::-DashboardController Premi::Front-End::Controllers::-HeaderController Premi::Front-End::Services::-AuthenticationService Premi::Front-End::Views::DashboardView
RFD33	Premi::Back-End::App::Controllers::-ErrorHandler

Requisito	Classi
	Premi::Back-End::App::Controllers::Errors::-PremiError Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Models::-ProjectModel Premi::Back-End::App::Routers::-ProjectRouter Premi::Front-End::Controllers::-ErrorMessageController Premi::Front-End::Directives::-premiErrorMessage Premi::Front-End::Model::ErrorInfo
RFD35	Premi::Back-End::App::Controllers::-ErrorHandler Premi::Back-End::App::Controllers::Errors::-PremiError Premi::Back-End::App::Controllers::-NotFoundHandler Premi::Back-End::App::Controllers::-ProjectController Premi::Back-End::App::Controllers::-Projects::NodeController Premi::Back-End::App::Controllers::-Projects::PathController Premi::Back-End::App::Controllers::-Projects::ProjectManagementController Premi::Back-End::App::Controllers::-StaticController Premi::Back-End::App::Controllers::-UserController Premi::Back-End::App::Controllers::Users::-AuthenticationController Premi::Back-End::App::Controllers::Users::-AuthorizationController Premi::Back-End::App::Routers::-ProjectRouter Premi::Back-End::App::Routers::-StaticRouter Premi::Back-End::App::Routers::-UserRouter Premi::Front-End::AppRun Premi::Front-End::Model::ErrorInfo
RFO36	Premi::Back-End::Config::Config

Requisito	Classi
	`Premi::Back-End::Server`

Tabella 3: Tracciamento Requisiti-Classi

5.3 Tracciamento Componenti-Requisiti

Componente	Requisiti
`Premi::Back-End`	RFO1
	RFO1.1
	RFO1.2.1
	RFO1.2.1.1
	RFO4
	RFD4.1
	RFO4.2
	RFO4.2.2
	RFO4.2.3
	RFO4.2.3.4
	RFO4.2.3.5
	RFO4.2.3.7
	RFD4.2.3.8
	RFD4.2.3.8.1
	RFD4.2.3.8.2
	RFO4.2.3.12
	RFO4.2.3.13
	RFD4.2.3.15
	RFD4.2.3.16
	RFO4.2.5
	RFO4.2.6
	RFO4.2.7
	RFD4.3
	RFD4.3.1
	RFD4.3.2
	RFD4.4
	RFD4.4.1
	RFD4.4.3
	RFD4.4.5
	RFD4.5
	RFF4.6
	RFF4.6.1
	RFF4.6.1.1
	RFF4.6.1.2
	RFF4.6.2
	RFO7
	RFD7.1
	RFO10
	RFO11
	RFO11.2
	RFO11.3
	RFD22
	RFD26
	RFO30.1

Componente	Requisiti
	RFO30.2 RFO30.3 RFO30.4 RFO32 RFD33 RFD35 RFO36
`Premi::Back-End::App`	RFO1 RFO1.1 RFO1.2.1 RFO1.2.1.1 RFO4 RFD4.1 RFO4.2 RFO4.2.2 RFO4.2.3 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.7 RFD4.2.3.8 RFD4.2.3.8.1 RFD4.2.3.8.2 RFO4.2.3.12 RFO4.2.3.13 RFD4.2.3.15 RFD4.2.3.16 RFO4.2.5 RFO4.2.6 RFO4.2.7 RFD4.3 RFD4.3.1 RFD4.3.2 RFD4.4 RFD4.4.1 RFD4.4.3 RFD4.4.5 RFD4.5 RFF4.6 RFF4.6.1 RFF4.6.1.1 RFF4.6.1.2 RFF4.6.2 RFO7 RFD7.1 RFO10 RFO11

Componente	Requisiti
	RFO11.2 RFO11.3 RFD22 RFD26 RFO30.1 RFO30.2 RFO30.3 RFO30.4 RFO32 RFD33 RFD35
`Premi::Back-End::App::Controllers`	RFO1 RFO1.1 RFO4 RFD4.1 RFO4.2 RFO4.2.2 RFO4.2.3 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.7 RFD4.2.3.8 RFD4.2.3.8.1 RFD4.2.3.8.2 RFO4.2.3.12 RFO4.2.3.13 RFD4.2.3.15 RFD4.2.3.16 RFO4.2.5 RFO4.2.6 RFO4.2.7 RFD4.3 RFD4.3.1 RFD4.3.2 RFD4.4 RFD4.4.1 RFD4.4.3 RFD4.4.5 RFD4.5 RFF4.6 RFF4.6.1 RFF4.6.1.1 RFF4.6.1.2 RFF4.6.2 RFO7 RFD7.1

Componente	Requisiti
	RFO10 RFO11 RFO11.2 RFO11.3 RFD22 RFD26 RFO30.1 RFO30.2 RFO30.3 RFO30.4 RFO32 RFD33 RFD35
`Premi::Back-End::App::Controllers::-Errors`	RFO30.3 RFO30.4 RFD33 RFD35
`Premi::Back-End::App::Controllers::-Projects`	RFO1 RFO1.1 RFO4 RFD4.1 RFO4.2 RFO4.2.3 RFO4.2.3.4 RFO4.2.3.5 RFD4.2.3.8 RFD4.2.3.8.1 RFD4.2.3.8.2 RFO4.2.3.12 RFO4.2.3.13 RFD4.2.3.15 RFD4.2.3.16 RFO4.2.5 RFO4.2.6 RFO4.2.7 RFD4.3 RFD4.3.1 RFD4.3.2 RFD4.4 RFD4.4.1 RFD4.4.3 RFD4.4.5 RFD4.5 RFF4.6

Componente	Requisiti
	RFF4.6.1 RFF4.6.1.1 RFF4.6.1.2 RFF4.6.2 RFO10 RFO11 RFO11.2 RFO11.3 RFD22 RFD33 RFD35
`Premi::Back-End::App::Controllers::-Users`	RFO30.1 RFO30.2 RFO30.3 RFO30.4 RFO32 RFD35
`Premi::Back-End::App::Models`	RFO1 RFO1.1 RFO1.2.1 RFO1.2.1.1 RFO4 RFD4.1 RFO4.2 RFO4.2.2 RFO4.2.3 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.7 RFD4.2.3.8 RFD4.2.3.8.1 RFD4.2.3.8.2 RFO4.2.3.12 RFO4.2.3.13 RFD4.2.3.15 RFD4.2.3.16 RFO4.2.5 RFO4.2.6 RFO4.2.7 RFD4.3 RFD4.3.1 RFD4.3.2 RFD4.4 RFD4.4.1 RFD4.4.3

Componente	Requisiti
	RFD4.4.5 RFD4.5 RFF4.6 RFF4.6.1 RFF4.6.1.1 RFF4.6.1.2 RFF4.6.2 RFO7 RFD7.1 RFO10 RFO11 RFO11.2 RFO11.3 RFD22 RFO30.1 RFO30.2 RFO30.3 RFO30.4 RFO32 RFD33
`Premi::Back-End::App::Routers`	RFO1 RFO1.1 RFO4 RFD4.1 RFO4.2 RFO4.2.2 RFO4.2.3 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.7 RFD4.2.3.8 RFD4.2.3.8.1 RFD4.2.3.8.2 RFO4.2.3.12 RFO4.2.3.13 RFD4.2.3.15 RFD4.2.3.16 RFO4.2.5 RFO4.2.6 RFO4.2.7 RFD4.3.1 RFD4.3.2 RFD4.4 RFD4.4.1 RFD4.4.3 RFD4.4.5

Componente	Requisiti
	RFD4.5 RFF4.6 RFF4.6.1 RFF4.6.1.1 RFF4.6.1.2 RFF4.6.2 RFO7 RFD7.1 RFO10 RFO11 RFO11.2 RFO11.3 RFD22 RFD26 RFO30.1 RFO30.2 RFO30.3 RFO30.4 RFO32 RFD33 RFD35
`Premi::Back-End::App::Views`	RFD26
`Premi::Back-End::Config`	RFO36
`Premi::Front-End`	RFO1 RFO1.1 RFO1.2 RFO2 RFD3 RFO4 RFD4.1 RFO4.2 RFO4.2.1 RFO4.2.2 RFO4.2.3 RFO4.2.3.1 RFO4.2.3.2 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.5.1 RFO4.2.3.5.2 RFO4.2.3.7 RFD4.2.3.9 RFD4.2.3.10 RFF4.2.3.11 RFO4.2.3.12 RFO4.2.3.13

Componente	Requisiti
	RFD4.2.3.15
	RFD4.2.3.16
	RFD4.2.3.17
	RFD4.2.3.18
	RFD4.2.3.19
	RFD4.2.3.20
	RFO4.2.5
	RFO4.2.6
	RFO4.2.7
	RFO4.2.8
	RFD4.3
	RFD4.3.1
	RFD4.3.2
	RFD4.3.3
	RFD4.4
	RFD4.4.1
	RFD4.4.3
	RFD4.4.5
	RFD4.5
	RFD4.5.1
	RFF4.6
	RFF4.6.1
	RFF4.6.1.1
	RFF4.6.1.2
	RFF4.6.2
	RFF4.6.3
	RFD4.7
	RFO5
	RFD6
	RFO7
	RFD7.1
	RFO7.2
	RFO7.3
	RFD7.4.1
	RFD7.4.2
	RFD7.5
	RFD7.6
	RFD7.6.1
	RFD7.6.2
	RFO7.7
	RFO8
	RFO9
	RFO10
	RFO10.1
	RFO10.2
	RFO11

Componente	Requisiti
	RFO11.1 RFO11.2 RFO11.3 RFD14 RFD14.1 RFD14.2 RFD14.3 RFD15 RFF15.1 RFF15.2 RFD15.3 RFO16 RFO16.1 RFF16.2 RFF16.3 RFD16.4 RFD17 RFO18 RFD22 RFD23 RFF25 RFF25.1 RFD26 RFO27 RFO30 RFO30.1 RFO30.1.1 RFO30.1.2 RFO30.1.3 RFO30.2 RFO30.2.1 RFO30.2.2 RFO30.2.3 RFO30.3 RFO30.4 RFO32 RFD33 RFD35
`Premi::Front-End::Controllers`	RFO1 RFO1.1 RFO1.2 RFO4 RFD4.1 RFO4.2 RFO4.2.1 RFO4.2.2

Componente	Requisiti
	RFO4.2.3
	RFO4.2.3.4
	RFO4.2.3.5
	RFO4.2.3.5.1
	RFO4.2.3.5.2
	RFO4.2.3.7
	RFD4.2.3.9
	RFD4.2.3.10
	RFF4.2.3.11
	RFO4.2.3.12
	RFO4.2.3.13
	RFD4.2.3.15
	RFD4.2.3.16
	RFD4.2.3.18
	RFD4.2.3.19
	RFD4.2.3.20
	RFO4.2.5
	RFO4.2.6
	RFO4.2.7
	RFD4.3
	RFD4.3.1
	RFD4.3.2
	RFD4.4
	RFD4.4.1
	RFD4.4.3
	RFD4.4.5
	RFD4.5
	RFD4.5.1
	RFF4.6
	RFF4.6.1
	RFF4.6.1.1
	RFF4.6.1.2
	RFF4.6.2
	RFF4.6.3
	RFD4.7
	RFO5
	RFO7
	RFD7.1
	RFO7.2
	RFO7.3
	RFD7.5
	RFD7.6
	RFD7.6.1
	RFD7.6.2
	RFO7.7
	RFO8

Componente	Requisiti
	RFO9 RFO10 RFO10.1 RFO10.2 RFO11 RFO11.1 RFO11.2 RFO11.3 RFD15 RFF15.1 RFF15.2 RFD15.3 RFO16 RFO16.1 RFF16.2 RFF16.3 RFD17 RFD22 RFF25 RFD26 RFO27 RFO30 RFO30.1 RFO30.1.1 RFO30.1.2 RFO30.1.3 RFO30.2 RFO30.2.1 RFO30.2.2 RFO30.2.3 RFO30.3 RFO30.4 RFO32 RFD33
`Premi::Front-End::Directives`	RFO4 RFD4.1 RFO4.2 RFO4.2.2 RFO4.2.3 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.5.1 RFO4.2.3.5.2 RFO4.2.3.7 RFD4.2.3.9 RFD4.2.3.10

Componente	Requisiti
	RFF4.2.3.11
	RFO4.2.3.12
	RFO4.2.3.13
	RFD4.2.3.15
	RFD4.2.3.16
	RFD4.2.3.18
	RFD4.2.3.19
	RFD4.2.3.20
	RFO4.2.5
	RFO4.2.6
	RFO4.2.7
	RFD4.3
	RFD4.3.1
	RFD4.3.2
	RFD4.3.3
	RFD4.4
	RFD4.4.1
	RFD4.4.3
	RFD4.4.5
	RFD4.5
	RFD4.5.1
	RFF4.6
	RFF4.6.1
	RFF4.6.1.1
	RFF4.6.1.2
	RFF4.6.2
	RFF4.6.3
	RFD4.7
	RFO7
	RFD7.1
	RFO7.2
	RFO7.3
	RFD7.6
	RFD7.6.1
	RFD7.6.2
	RFO7.7
	RFO10
	RFO10.1
	RFO10.2
	RFO11
	RFO11.1
	RFO11.2
	RFO11.3
	RFD15
	RFF15.1
	RFF15.2

Componente	Requisiti
	RFD15.3 RFO16 RFO16.1 RFF16.2 RFF16.3 RFD22 RFF25 RFO27 RFO30.3 RFO30.4 RFD33
`Premi::Front-End::Model`	RFO4 RFD4.1 RFO4.2 RFO4.2.3 RFO4.2.3.1 RFO4.2.3.2 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.5.1 RFO4.2.3.5.2 RFO4.2.3.7 RFD4.2.3.15 RFD4.2.3.16 RFD4.2.3.17 RFD4.2.3.18 RFD4.2.3.19 RFD4.2.3.20 RFD4.4 RFD4.4.1 RFD4.4.3 RFD4.4.5 RFF4.6 RFF4.6.1 RFF4.6.1.1 RFF4.6.1.2 RFF4.6.2 RFD4.7 RFO5 RFD6 RFO7 RFO7.2 RFO7.3 RFO7.7 RFD22 RFO30.3

Componente	Requisiti
	RFO30.4 RFD33 RFD35
`Premi::Front-End::Services`	RFO1 RFO1.1 RFO1.2 RFO2 RFD3 RFO4 RFD4.1 RFO4.2 RFO4.2.1 RFO4.2.2 RFO4.2.3 RFO4.2.3.4 RFO4.2.3.5 RFO4.2.3.5.1 RFO4.2.3.5.2 RFO4.2.3.7 RFD4.2.3.15 RFD4.2.3.16 RFD4.2.3.18 RFD4.2.3.19 RFD4.2.3.20 RFO4.2.5 RFO4.2.6 RFO4.2.7 RFO4.2.8 RFD4.3 RFD4.3.1 RFD4.3.3 RFD4.4 RFD4.4.1 RFD4.4.3 RFD4.4.5 RFD4.5 RFD4.5.1 RFF4.6 RFF4.6.3 RFD4.7 RFO5 RFD6 RFO7 RFD7.1 RFD7.4.1 RFD7.4.2

Componente	Requisiti
	RFD7.6.2 RFO7.7 RFO8 RFO9 RFO10 RFO10.1 RFO10.2 RFO11 RFO11.1 RFO11.2 RFO11.3 RFD16.4 RFD22 RFD23 RFF25.1 RFO30 RFO30.1 RFO30.1.1 RFO30.1.2 RFO30.1.3 RFO30.2 RFO30.2.1 RFO30.2.2 RFO30.2.3 RFO32
`Premi::Front-End::Views`	RFO1 RFO1.1 RFO1.2 RFO4.2 RFO4.2.1 RFD4.4 RFO5 RFO7 RFD7.1 RFO7.2 RFO7.3 RFD7.5 RFD7.6 RFD7.6.1 RFD7.6.2 RFO8 RFO9 RFD14 RFD14.1 RFD14.2 RFD14.3

Componente	Requisiti
	RFO16 RFO16.1 RFF16.2 RFF16.3 RFD17 RFO18 RFD22 RFD26 RFO27 RFO30 RFO30.1 RFO30.1.1 RFO30.1.2 RFO30.1.3 RFO30.2 RFO30.2.1 RFO30.2.2 RFO30.2.3 RFO32

Tabella 4: Tracciamento Componenti-Requisiti

5.4 Tracciamento Requisiti-Componenti

Requisito	Componenti
RFO1	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Services Premi::Front-End::Views
RFO1.1	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Services Premi::Front-End::Views
RFO1.2	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Services Premi::Front-End::Views
RFO1.2.1	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Models
RFO1.2.1.1	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Models
RFO2	Premi::Front-End Premi::Front-End::Services
RFD3	Premi::Front-End Premi::Front-End::Services
RFO4	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End

Requisito	Componenti
	Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services
RFD4.1	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services
RFO4.2	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services Premi::Front-End::Views
RFO4.2.1	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Services Premi::Front-End::Views
RFO4.2.2	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Services
RFO4.2.3	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers

Requisito	Componenti
	Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services
RFO4.2.3.1	Premi::Front-End Premi::Front-End::Model
RFO4.2.3.2	Premi::Front-End Premi::Front-End::Model
RFO4.2.3.4	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services
RFO4.2.3.5	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services
RFO4.2.3.5.1	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services
RFO4.2.3.5.2	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model

Requisito	Componenti
	Premi::Front-End::Services
RFO4.2.3.7	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services
RFD4.2.3.8	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers
RFD4.2.3.8.1	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers
RFD4.2.3.8.2	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers
RFD4.2.3.9	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives
RFD4.2.3.10	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives
RFF4.2.3.11	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives
RFO4.2.3.12	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects

Requisito	Componenti
	Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives
RFO4.2.3.13	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives
RFD4.2.3.15	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services
RFD4.2.3.16	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services
RFD4.2.3.17	Premi::Front-End Premi::Front-End::Model
RFD4.2.3.18	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services

Requisito	Componenti
RFD4.2.3.19	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services
RFD4.2.3.20	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services
RFO4.2.5	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Services
RFO4.2.6	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Services
RFO4.2.7	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Services
RFO4.2.8	Premi::Front-End Premi::Front-End::Services
RFD4.3	Premi::Back-End

Requisito	Componenti
	Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Services
RFD4.3.1	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Services
RFD4.3.2	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives
RFD4.3.3	Premi::Front-End Premi::Front-End::Directives Premi::Front-End::Services
RFD4.4	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services Premi::Front-End::Views

Requisito	Componenti
RFD4.4.1	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services
RFD4.4.3	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services
RFD4.4.5	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services
RFD4.5	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives

Requisito	Componenti
	Premi::Front-End::Services
RFD4.5.1	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Services
RFF4.6	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services
RFF4.6.1	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model
RFF4.6.1.1	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model
RFF4.6.1.2	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers

Requisito	Componenti
	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model
RFF4.6.2	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model
RFF4.6.3	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Services
RFD4.7	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services
RFO5	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Model Premi::Front-End::Services Premi::Front-End::Views
RFD6	Premi::Front-End Premi::Front-End::Model Premi::Front-End::Services
RFO7	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services Premi::Front-End::Views
RFD7.1	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers

Requisito	Componenti
	Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Services Premi::Front-End::Views
RFO7.2	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Views
RFO7.3	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Views
RFD7.4.1	Premi::Front-End Premi::Front-End::Services
RFD7.4.2	Premi::Front-End Premi::Front-End::Services
RFD7.5	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Views
RFD7.6	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Views
RFD7.6.1	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Views
RFD7.6.2	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Services Premi::Front-End::Views
RFO7.7	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services
RFO8	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Services Premi::Front-End::Views

Requisito	Componenti
RFO9	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Services Premi::Front-End::Views
RFO10	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Services
RFO10.1	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Services
RFO10.2	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Services
RFO11	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Services
RFO11.1	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Services
RFO11.2	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End

Requisito	Componenti
	Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Services
RFO11.3	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Services
RFD14	Premi::Front-End Premi::Front-End::Views
RFD14.1	Premi::Front-End Premi::Front-End::Views
RFD14.2	Premi::Front-End Premi::Front-End::Views
RFD14.3	Premi::Front-End Premi::Front-End::Views
RFD15	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives
RFF15.1	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives
RFF15.2	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives
RFD15.3	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives
RFO16	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Views
RFO16.1	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Views
RFF16.2	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives

Requisito	Componenti
	Premi::Front-End::Views
RFF16.3	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Views
RFD16.4	Premi::Front-End Premi::Front-End::Services
RFD17	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Views
RFO18	Premi::Front-End Premi::Front-End::Views
RFD22	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model Premi::Front-End::Services Premi::Front-End::Views
RFD23	Premi::Front-End Premi::Front-End::Services
RFF25	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives
RFF25.1	Premi::Front-End Premi::Front-End::Services
RFD26	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Routers Premi::Back-End::App::Views Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Views
RFO27	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Views
RFO30	Premi::Front-End Premi::Front-End::Controllers

Requisito	Componenti
	Premi::Front-End::Services Premi::Front-End::Views
RFO30.1	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Users Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Services Premi::Front-End::Views
RFO30.1.1	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Services Premi::Front-End::Views
RFO30.1.2	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Services Premi::Front-End::Views
RFO30.1.3	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Services Premi::Front-End::Views
RFO30.2	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Users Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Services Premi::Front-End::Views
RFO30.2.1	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Services Premi::Front-End::Views
RFO30.2.2	Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Services Premi::Front-End::Views
RFO30.2.3	Premi::Front-End Premi::Front-End::Controllers

Requisito	Componenti
	Premi::Front-End::Services Premi::Front-End::Views
RFO30.3	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Errors Premi::Back-End::App::Controllers::-Users Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model
RFO30.4	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Errors Premi::Back-End::App::Controllers::-Users Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model
RFO32	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Users Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Services Premi::Front-End::Views
RFD33	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Errors Premi::Back-End::App::Controllers::-Projects

Requisito	Componenti
	Premi::Back-End::App::Models Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Controllers Premi::Front-End::Directives Premi::Front-End::Model
RFD35	Premi::Back-End Premi::Back-End::App Premi::Back-End::App::Controllers Premi::Back-End::App::Controllers::-Errors Premi::Back-End::App::Controllers::-Projects Premi::Back-End::App::Controllers::-Users Premi::Back-End::App::Routers Premi::Front-End Premi::Front-End::Model
RFO36	Premi::Back-End Premi::Back-End::Config

Tabella 5: Tracciamento Requisiti-Componenti

Appendice A Interfaccia REST

L'interfaccia per il *back-end*$_G$ del *progetto*$_G$ Premi è realizzata in stile *REST-like*$_G$, basato sullo stile *REST*$_G$ ma che non ne segue strettamente tutti i principi. Questo consente di effettuare l'autenticazione dell'utente salvando un cookie e memorizzando nel *server*$_G$ una sessione associata al *client*$_G$. Una volta creata sessione, l'interfaccia con cui si possono accedere e modificare i dati nel database può considerarsi effettivamente *REST*$_G$.
I dati scambiati mediante l'interfaccia sono rappresentati con il formato JSON, che si integra facilmente con le tecnologie ed il linguaggio utilizzati per sviluppare Premi.
Se lo scambio di dati avviene correttamente il *server*$_G$ può fornire in risposta un messaggio di conferma:

```
1 {
2 "status" : "ok"
3 }
```

A.1 Errori

In caso di errori il *server*$_G$ risponde con un messaggio d'errore in formato JSON, definito secondo lo schema:

```
1 {
2 "code": [codice dell'errore],
3 "title": [titolo dell'errore],
4 "message": [descrizione testuale dell'errore]
5 }
```

A.1.1 Errori generici

Gli errori generici sono gestiti dalla classe `ErrorHandler` ad eccezione dell'errore `404` che viene gestito da `NotFoundHandler`. In questo modo si ottiene una gestione degli errori più flessibile e modulare, in quanto è possibile inserire in coda allo stack di chiamate di ogni route la classe `NotFountHandler`, riducendo così la complessità della classe `ErrorHandler`.

A.1.1.1 Errori lato server

Nel caso si verifichi un errore lato server, questo risponde con un codice HTTP del tipo `5xx`, di seguito vengono specificati gli errori che possono essere sollevati:

- `500 Errore sconosciuto`: il messaggio descrive un errore generico del server che avviene quando si verifica una condizione non gestibile e quindi non identificabile con un errore specifico.

A.1.1.2 Errori nelle richieste da parte dei client

Nel caso si verifichi un errore riguardo le richieste ricevute dal client, il server risponde con un codice HTTP del tipo `4xx`, di seguito vengono specificati gli errori che possono essere sollevati:

- `400`: il server non può gestire la richiesta in seguito ad una generica richiesta errata. Il contenuto del messaggio d'errore varia in base alla tipologia di richiesta ricevuta;

- `401`: Accesso non autorizzato;
- `404`: Pagina non trovata.

A.1.2 Errori specifici di Premi

Questi errori rappresentano situazioni specifiche del sistema Premi, sono quindi stati definiti dei codici personalizzati per questa tipologia di errori basandosi sull'idea dei codici d'errore HTTP standard. I codici dei messaggi d'errore sono stati assegnati secondo diverse categorie, in modo da poter individuare facilmente quale componente dell'applicazione ha causato l'errore. Ad ogni categoria è stato assegnato successivamente un intervallo di codici possibili, evitato di utilizzare gli intervalli `4xxx` e `5xxx` per non creare ambiguità con i codici d'errore standard descritti nella sezione precedente.
Le categorie di errori definite sono le seguenti:

- `1xxx`: errori di autenticazione e del middleware `userById`;
- `2xxx`: errori del middleware `projectById`;
- `3xxx`: errori del middleware `nodeById`;
- `6xxx`: errori del middleware `associationById`;
- `7xxx`: errori del middleware `pathById`;
- `8xxx`: errori del controller `ProjectManagementController`;
- `9xxx`: errori del controller `NodeController`;
- `10xxx`: errori del controller `PathController`;
- `11xxx`: errori degli oggetti `ProjectModel`;
- `12xxx`: errori degli oggetti `PathModel`.

Gli errori specifici di Premi vengono gestiti dalla classe `PremiError` e sono di seguito elencati sotto forma di:

`Codice Titolo`: messaggio d'errore

- `1000 Utente non trovato`: l'identificativo utente fornito non è un identificativo valido;
- `1002 Credenziali non valide`: è necessario fornire un indirizzo email ed una password valide;
- `2000 Progetto non trovato`: l'identificativo del progetto fornito non è un identificativo valido;
- `3000 Nodo non trovato`: l'identificativo del nodo fornito non è un identificativo valido;
- `6000 Associazione non trovata`: l'identificativo dell'associazione fornita non è un identificativo valido;

- `7000 Percorso non trovato`: l'identificativo del percorso fornito non è un identificativo valido;
- `8000 Progetto corrotto`: errore nella ricerca dei campi dati relativi al progetto indicato;
- `8001 Dati non validi`: i dati relativi al progetto sono vuoti o errati;
- `8002 Progetto corrotto`: errore durante l'eliminazione del progetto;
- `9000 Dati non validi`: i dati relativi al contenuto del nodo non sono validi o sono formattati in modo errato;
- `9001 Nodi non validi`: gli identificativi dei nodi forniti non sono validi oppure non esistono oppure il nodo entrante coincide con il nodo uscente;
- `10000 Dati non validi`: i dati per la modifica del percorso non sono definiti;
- `11001 Nodo del progetto non trovato`: il nodo riferito nella relazione o nel percorso non è stato trovato all'interno del progetto indicato;
- `11002 Nodo padre non esistente`: il nodo padre indicato non esiste o non corrisponde ad un nodo valido;
- `11003 Nodo non valido`: impossibile eliminare il nodo radice della mappa mentale.
- `12000 Nodo del percorso non trovato`: il nodo riferito non è stato trovato all'interno del percorso indicato.

A.2 Risorse REST

In seguito vengono riportate le risorse $REST_G$ fornite, associate al tipo di metodo HTTP che è possibile richiedere su di esse e ai permessi necessari per effettuare la richiesta. In particolare, i permessi sono:

- **Utente**: la risorsa può essere acceduta da qualsiasi tipo di utente;
- **Utente autenticato**: la risorsa può essere acceduta solo dagli utenti che hanno effettuato il login.

Inoltre, per ogni risorsa, sono stati specificati i formati per lo scambio dei dati in JSON:

- **Request:** rappresenta l'oggetto JSON che dovrà essere passato alla risorsa $REST_G$;
- **Response:** rappresenta l'oggetto JSON che fornirà in risposta la risorsa $REST_G$.

- `/signup`
  - **Method:** POST;
  - **Livello di Accesso:** utente;

- **Descrizione:** crea un nuovo account. L'indirizzo email rappresenta lo username dell'utente e lo identifica univocamente, pertanto un utente non può registrare più account con lo stesso indirizzo di posta elettronica. Restituisce un messaggio di conferma se viene effettuato correttamente, altrimenti un errore;
- **Request:** lo scambio dei dati dell'utente avviene attraverso una form che deve avere i seguenti campi:

```
1 {
2 "username" : [username che corrisponde all'indirizzo di
     posta elettronica con cui ci si vuole registrare];
3 "password" : [la password per l'account con cui ci si vuole
     registrare].
4 }
```

- `/login`
  - **Method:** POST;
  - **Livello di Accesso:** utente;
  - **Descrizione:** effettua l'accesso all'account. Restituisce un messaggio di conferma se viene effettuato correttamente, altrimenti un errore;
  - **Request:** lo scambio dei dati dell'utente avviene attraverso una form che deve avere i seguenti campi:

```
1 {
2 "username" : [username che corrisponde all'indirizzo di
     posta elettronica che identifica l'utente];
3 "password" : [la password dell'account dell'utente].
4 }
```

- `/logout`
  - **Method:** GET;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** effettua il logout. Restituisce un messaggio di conferma se viene effettuato correttamente, altrimenti un errore;
- `/projects`
  - **Method:** POST;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** crea un nuovo *progetto*$_G$ relativo all'utente della sessione corrente e utilizzando il nome specificato nei parametri POST. restituisce un oggetto JSON contenente l'id del progetto, le indicazioni per il layout, le liste di nodi, le relazioni e le associazioni della *mappa mentale*$_G$ del *progetto*$_G$ `:projectId`, altrimenti un messaggio di errore;
  - **Request:**

```
1 {
2 "name": [nome del progetto]
3 }
```

- **Response:**

```
{
"_id" : [identificativo del progetto],
"name" : [nome del progetto],
"fontFamily" : [nome della famiglia],
"fontColor" : [nome del colore],
"bkgColor" : [nome del colore],
"relations" : [array JSON di relazioni, con attributi
   specificati nei diagrammi delle classi],
"nodes" : [array JSON di nodi, con attributi specificati nei
    diagrammi delle classi]
}
```

- `/projects`
  - **Method:** GET;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** restituisce un array di oggetti JSON che descrive identificativo e nome dei *progetti*$_G$ nel database per l'utente della sessione. Inoltre, per ogni *progetto*$_G$, fornisce l'elenco dei *percorsi di presentazione*$_G$ associati;
  - **Response:**

```
{
[{
"_id" : [identificativo del progetto],
"name": [nome del progetto],
"paths": [
{
"_id": [identificativo del precorso],
"name" : [nome del percorso]
}
]
}]
}
```

- `/projects/:projectId`
  - **Method:** GET;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** restituisce un oggetto JSON contenente le indicazioni per il layout, le liste di nodi, le relazioni e le associazioni della *mappa mentale*$_G$ del *progetto*$_G$ `:projectId`, altrimenti un messaggio di errore;
  - **Response:**

```
{
"name" : [nome del progetto],
"fontFamily" : [nome della famiglia],
"fontColor" : [nome del colore],
"bkgColor" : [nome del colore],
```

```
6 "relations" : [array JSON di relazioni, con attributi
     specificati nei diagrammi delle classi],
7 "nodes" : [array JSON di nodi, con attributi specificati nei
      diagrammi delle classi]
8 }
```

- `/projects/:projectId`
  - **Method:** PUT;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** modifica le impostazioni del *progetto*$_G$ `:projectId`, relativo all'utente della sessione corrente. Restituisce un messaggio di conferma se viene effettuato correttamente, altrimenti un errore;
  - **Request:**

```
1 {
2 "name" : [nome del progetto],
3 "bkgColor" : [colore dello sfondo],
4 "fontColor" : [colore per i testi],
5 "fontFamily" : [famiglia del font per i testi]
6 }
```

- `/projects/:projectId`
  - **Method:** DELETE;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** rimuove il *progetto*$_G$ con identificativo `:projectId` relativo all'utente della sessione corrente dal database. Restituisce un messaggio di conferma, altrimenti uno di errore;
- `/projects/:projectId/presentations`
  - **Method:** GET;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** restituisce tutti i nodi della mappa mentale, quindi del *progetto*$_G$ `projectId` dell'utente della sessione. Restituisce un oggetto JSON creato ad hoc per eseguire una presentazione nel font-end dell'applicazione. Altrimenti restituisce un messaggio di errore;
  - **Response:**

```
1 {
2 "frames" : [
3 {
4 "node" : [contenuto e informazioni del nodo, con attributi
     definiti nei diagrammi delle classi],
5 "family" : [{
6 {
7 "childern":[
8 {
```

```
9  "id" : ["identificativo dei nodi figli"],
10 "title": ["titolo del nodo"]
11 }]
12 "parent" : {
13 "id" : ["identificativo del nodo padre"],
14 "title": ["titolo del nodo"]
15 }
16 }],
17 "associations" : [
18 {
19 "id" : [identificativo dei nodi con cui esiste una
      associazione],
20 "title": ["titolo del nodo"]
21 }
22 ]
23 }
24 ]
25 }
```

- `/projects/:projectId/nodes/:nodeId`
  - **Method:** POST;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** inserisce nel database un nuovo nodo vuoto come figlio del nodo `:nodeId`, relativo al *progetto*$_G$ `:projectId`. Restituisce i dati del nuovo nodo oppure un messaggio di errore;
  - **Response:**

```
1  {
2  "nodeId" : [identificativo del nodo appena creato]
3  "contents" : [
4  {
5  "content" : [testo o URL],
6  "x" : [coordinata sull'asse x dell'elemento nel frame],
7  "y" : [coordinata sull'asse y dell'elemento nel frame],
8  "height" : [altezza dell'elemento],
9  "width" : [larghezza dell'elemento],
10 "class" : [title, text o imgURL]
11 }
12 ]
13 }
```

- `/projects/:projectId/nodes/:nodeId`
  - **Method:** PUT;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** aggiorna il contenuto del nodo `:nodeId`, nel *progetto*$_G$ `:projectId`, relativo all'utente della sessione. Restituisce un messaggio di conferma oppure un messaggio di errore;
  - **Request:**

```
{
"contents" : [
{
"content" : [testo o URL],
"x" : [coordinata sull'asse x dell'elemento nel frame],
"y" : [coordinata sull'asse y dell'elemento nel frame],
"height" : [altezza dell'elemento],
"width" : [larghezza dell'elemento],
"class" : [title, text o imgURL]
}
]
}
```

- `/projects/:projectId/nodes/:nodeId`
  - **Method:** DELETE;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** elimina dal database il nodo `:nodeId`, nel *progetto*$_G$ `:projectId`, relativo all'utente della sessione. Oltre ad esso viene eliminato l'eventuale sotto-albero di cui `:nodeId` è padre, le associazioni che coinvolgono i nodi del sotto-albero e vengono aggiornati i *percorsi*$_G$ in cui il nodo compariva. Restituisce la nuova mappa mentale, con nodi e relazioni aggiornati oppure un messaggio di errore;
  - **Response:**

```
{
"nodes" : [array JSON di nodi, con attributi specificati nei
    diagrammi delle classi],
"relations" : [array JSON di relazioni, con attributi
   specificati nei diagrammi delle classi]
}
```

- `/projects/:projectId/associations`
  - **Method:** POST;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** crea una relazione di tipo associazione nel *progetto*$_G$ `:projectId`, relativo all'utente della sessione. Richiede gli identificativi dei nodi che formano l'associazione. Restituisce un messaggio di conferma oppure un messaggio di errore;
  - **Request:**

```
{
"sourceId" : [identificativo del nodo sorgente della
   relazione],
"destinationId" : [identificativo del nodo destinazione
   della relazione]
}
```

- **Response:**

```
1 {
2 "associationId" : [identificativo dell'associazione creata]
3 }
```

- `/projects/:projectId/associations/:associationId`
  - **Method:** DELETE;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** elimina una relazione di tipo associazione nel *progetto*$_G$ `:projectId`, relativo all'utente della sessione. Richiede gli identificativi dei nodi che formano l'associazione. Restituisce un messaggio di conferma oppure un messaggio di errore;
  - **Request:**

```
1 {
2 "associationId" : [identificativo dell'associazione creata]
3 }
```

- `/projects/:projectId/paths`
  - **Method:** POST;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** aggiunge un *percorso di presentazione*$_G$ vuoto al *progetto*$_G$ `:projectId` relativo all'utente della sessione. Richiede che venga specificato un nome per il *percorso*$_G$. Restituisce l'identificativo del nuovo *percorso*$_G$, generato automaticamente dal database o un messaggio di errore;
  - **Request:**

```
1 {
2 "name" : [nome per il percorso]
3 }
```

  - **Response:**

```
1 {
2 "pathId" : [identificativo del percorso],
3 "name" : [nome del percorso]
4 }
```

- `/projects/:projectId/paths`
  - **Method:** GET;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** restituisce i *percorsi di presentazione*$_G$ del *progetto*$_G$ `:projectId` relativo all'utente della sessione. Altrimenti restituisce un messaggio di errore;
  - **Response:**

```
{
"paths" : [{
"id" : [identificativo del percorso],
"name" : [nome del percorso]
}]
}
```

- `/projects/:projectId/paths/:pathId/:nodeId`
  - **Method:** POST;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** aggiunge il nodo `:nodeId` al *percorso*$_G$ `:pathId` relativo al *progetto*$_G$ `projectId` dell'utente della sessione. Richiede che venga specificato l'identificativo del nodo da aggiungere. Restituisce tutti i nodi del *percorso*$_G$ `:pathId` relativi al *progetto*$_G$ `projectId` dell'utente della sessione oppure un messaggio di errore;
  - **Response:**

```
{
"name": [nome del percorso]
"nodes" : [array JSON di nodi, con attributi specificati nei
    diagrammi delle classi]
}
```

- `/projects/:projectId/paths/:pathId`
  - **Method:** GET;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** restituisce tutti i nodi del *percorso*$_G$ `:pathId` relativi al *progetto*$_G$ `projectId` dell'utente della sessione oppure un messaggio di errore;
  - **Response:**

```
{
"name": [nome del percorso]
"nodes" : [array JSON di nodi, con attributi specificati nei
    diagrammi delle classi]
}
```

- `/projects/:projectId/paths/:pathId`
  - **Method:** PUT;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** aggiorna le informazioni del *percorso*$_G$ `:pathId` nel *progetto*$_G$ `:projectId`, relativo all'utente della sessione, in particolare consente di rinominarlo. Richiede che venga specificato un nuovo nome per il *percorso*$_G$. Restituisce un messaggio di conferma o un messaggio di errore;
  - **Request:**

```
1 {
2 "name" : [nome per il percorso]
3 }
```

- `/projects/:projectId/paths/:pathId`
  - **Method:** DELETE;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** rimuove il *percorso di presentazione*$_G$ `:pathId` dal *progetto*$_G$ `:projectId`, relativo all'utente della sessione. Restituisce un messaggio di conferma oppure un messaggio di errore;
- `/projects/:projectId/paths/:pathId/:nodeId`
  - **Method:** DELETE;
  - **Livello di Accesso:** utente autenticato;
  - **Descrizione:** rimuove il nodo `:nodeId` dal *percorso*$_G$ `:pathId`, relativo al *progetto*$_G$ `projectId` dell'utente della sessione. Restituisce tutti i nodi del *percorso*$_G$ `:pathId` relativi al *progetto*$_G$ `projectId` dell'utente della sessione oppure un messaggio di errore;
  - **Response:**

```
1 {
2 "name": [nome del percorso]
3 "nodes" : [array JSON di nodi, con attributi specificati nei
      diagrammi delle classi]
4 }
```

- `/statics/userManual`
  - **Method:** GET;
  - **Livello di Accesso:** utente;
  - **Descrizione:** fornisce al chiamante l'URL della pagina *web*$_G$ contenente il manuale utente dell'applicativo;

Appendice B Design patterns

B.1 Chain of Responsibility

Scopo Evitare l'accoppiamento tra il mittente di una richiesta ed il destinatario, in modo che più di un singolo oggetto possa eseguire la richiesta. Concatenare gli oggetti destinatari e passare la richiesta di oggetto in oggetto, finché uno di questi non riesce ad esaudirla.

Motivazione L'oggetto che ha iniziato la richiesta non è a conoscenza di chi la esaudirà, per questo si parla di destinatario implicito. Ogni oggetto della catena condivide un' interfaccia comune *Handler* per la gestione delle richieste e per accedere al successivo elemento della catena. Questo per consentire il passaggio lungo la catena e per assicurare che l'oggetto che eventualmente esaudirà la richiesta rimanga implicito. Il metodo che viene mantenuto in tutte le interfacce per creare la catena si chiama *handle*. *handle*, nella versione standard, esegue la chiamata al successore nella catena. Alla fine della catena viene fatto l'overriding di questo metodo, implementando la richiesta iniziale oppure gestendo l'errore.

Applicabilità *Chain of Responsibility* si usa nei seguenti casi:

- Più di un oggetto può gestire la richiesta ed il ricevente che la gestirà non è conosciuto a priori;
- Si vuole passare una richiesta ad uno dei molti oggetti, senza esplicitare il ricevente;
- L'insieme di oggetti che gestirà una richiesta deve essere definito dinamicamente.

Utilizzo Express usa *Chain of Responsibility* per la gestione dei middleware e del routing. Nell'architettura viene utilizzato all'interno del package *Back-End*$_G$::App. Ogni *ConcreteHandler* eredita da *MiddlewareHandler*, che corrisponde all'interfaccia astratta *Handler* del Design Pattern. Con Express i middleware corrispondono ai *ConcreteHandler*. L'implementazione di questi risulta sotto alcuni aspetti differente da una normale implementazione del pattern:

1. I middleware di Express possono essere delle classi che implementano il metodo *handle* oppure delle funzioni secondo lo stile funzionale delle *librerie*$_G$ e dei moduli di Node.js. Nella nostra architettura abbiamo utilizzato principalmente la seconda versione;
2. Nel *Design Pattern*$_G$ è previsto che l'oggetto *ConcreteHandler* abbia un riferimento (successor) al *ConcreteHandler* successivo. Express, anzichè un riferimento, passa al metodo che esegue il middleware una *callback*$_G$. Il middleware che esegue la *callback*$_G$ passa il controllo all'oggetto del *server*$_G$ di Express, il quale passerà a sua volta il controllo al successivo middleware.

Express divide i middleware in due tipologie:

1. *Middleware standard* con 3 parametri formali;
2. *Middleware per la gestione degli errori* con 4 parametri formali, ovvero i 3 del middleware standard più un parametro per gli errori.

Ogni middleware può passare il controllo al middleware standard successivo, oppure ad un middleware per la gestione degli errori, passandogli l'errore relativo. Ogni middleware di Express deve essere invocato con i parametri elencati nel seguente ordine:

1. L'eventuale errore da gestire, in caso si tratti di un middleware per la gestione degli errori;
2. L'oggetto contenente la richiesta da risolvere;
3. L'oggetto che conterrà la risposta;
4. La *callback*$_G$ da utilizzare per passare il controllo al successivo middleware.

Struttura

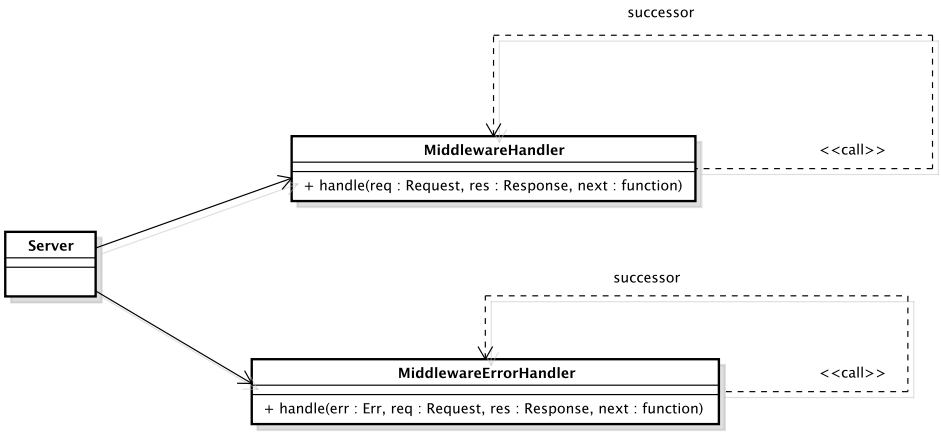


Figura 122: Chain Of Responsibility contestualizzato

Collaborazioni Quando un *client*$_G$ effettua una richiesta, questa viene propagata lungo la catena finché un *ConcreteHandler* non si assume la responsabilità di gestirla.

Svantaggi È necessario porre attenzione a come si configura la catena. Infatti, in caso di configurazione inappropriata, sussiste il rischio che la richiesta passi per diversi punti della catena senza essere mai realizzata, oppure che la richiesta non non venga realizzata perché non esiste un *ConcreteHandler* che la possa risolvere.

B.2 Dependecy Injection

Scopo Separare il codice della componente dal codice che si occupa di risolvere le dipendenze con le altre componenti.

Motivazione Lasciare al componente il compito di risolvere le proprie dipendenze, creando gli oggetti necessari al suo funzionamento, aumenta l'accoppiamento tra le componenti e rende più difficoltoso progettare i test di unità.
Con questo pattern invece è possibile esprimere le dipendenze in modo dichiarativo e utilizzare un oggetto *contenitore* per risolverle dinamicamente a *runtime*$_G$. In questo modo è possibile scegliere anche quale componente iniettare in base allo stato del programma.

Applicabilità Questo pattern viene utilizzato dalla maggior parte dei *framework*$_G$ moderni. In particolare, *AngularJS*$_G$ offre il servizio `$injector` che permette di invocare delle funzioni iniettando al loro interno degli oggetti.

Struttura I componenti coinvolti nel *Dependency Injection* sono:

- Un client$_G$ che viene creato e riceve le dipendenze;
- Un *contenitore* che si occupa di creare il *client*$_G$ e di iniettarvi le dipendenze;
- Un *servizio* che deve essere iniettato al *client*$_G$.

Nello specifico di *AngularJS*$_G$ `$injector` funziona da *contenitore* che si occupa di risolvere le dipendenze. I client$_G$ sono rappresentati dalle funzioni che costruiscono i componenti dell'applicazione, tipicamente *controller*$_G$ o service. Il *servizio* è un oggetto service che può essere definito dall'utente oppure uno di quelli resi disponibili da *AngularJS*$_G$.

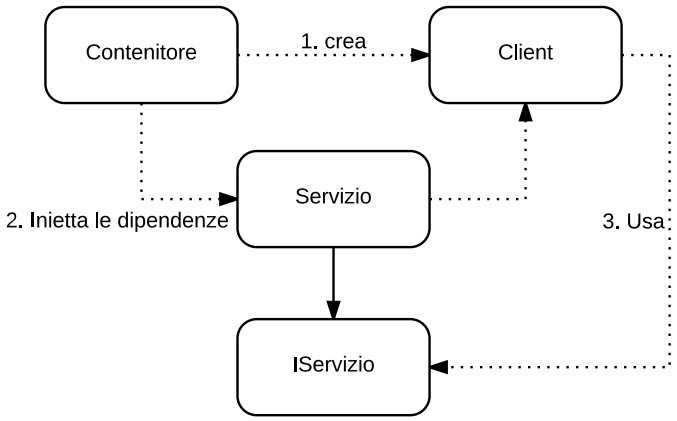


Figura 123: Struttura logica del Dependency Injection

L'*injection* delle dipendenze può essere fatta in due modi:

- **Constructor Injection**: le dipendenze vengono dichiarate come parametri del costruttore che il *container* andrà a valorizzare quando crea l'oggetto. In questo modo l'oggetto costruito è subito utilizzabile e non è mai in uno stato inconsistente. Tuttavia si rende necessario avere dei costruttori con un elevato numero di parametri, che potrebbero rendere i costruttori difficili da comprendere. *AngularJS*$_G$ usa questo tipo di *dependency injection*;
- **Setter Injection**: le dipendenze vengono dichiarate come metodi *setter* del componente. Questa soluzione è meno ambigua, funziona bene con le gerarchie di classi e non soffre del *telescoping* dei costruttori. Non è però possibile avere oggetti immutabili. È possibile che la componente rimanga in uno stato inconsistente, in quanto deve essere costruita per passi.

Svantaggi Eventuali errori legati alla risoluzione delle dipendenze o alla loro implementazione vengono rilevati solamente a *runtime*$_G$.

B.3 Facade

Scopo Fornire un'interfaccia unica per un sottosistema complesso. Rendere una *libreria*$_G$ più facile da capire, usare e testare. Diminuire le dipendenze tra sottosistemi senza nascondere le funzionalità di basso livello.

Motivazione Quando un sistema complesso viene strutturato in sottosistemi, le dipendenze tra essi possono aumentare. Applicare il pattern *Facade*$_G$ in un sottosistema aiuta a diminuire le dipendenze dall'esterno verso le classi che lo compongono. Il sottosistema possederà un'interfaccia semplificata che il *client*$_G$ può utilizzare anziché dover gestire numerosi oggetti. Utilizzare il pattern *Facade*$_G$ promuove un accoppiamento debole tra un sottosistema ed i *client*$_G$, che comporta una maggiore flessibilità nello sviluppo: è possibile modificare il sottosistema senza che i *client*$_G$ debbano adeguarsi a loro volta.
Ciononostante i *client*$_G$ possono comunque accedere alle funzionalità di basso livello ed utilizzare le classi del sottosistema.

Applicabilità Facade$_G$ si usa nei seguenti casi:

- Si vuole fornire una singola interfaccia semplice per un sottosistema complesso;
- Si vuole promuovere il disaccoppiamento tra sottosistemi e *client*$_G$, semplificando le dipendenze;
- Si vuole stratificare un sistema: è possibile definire una classe *Facade*$_G$ come punto d'ingresso per ogni livello di sottosistema. In questo modo, se vi sono dipendenze fra sottosistemi, essi possono comunicare fra loro attraverso la propria *Facade*$_G$.

Utilizzo Nel package `Premi::Back-End::App::Models` è stata definita una classe *Facade*$_G$ `Premi::Back-End::App::Models::ProjectModel`. Essa rappresenta un *progetto*$_G$ contenente nodi, relazioni e *percorsi di presentazione*$_G$ e fornisce un'interfaccia per le classi del package che compongono la mappa mentale. I *controller*$_G$ non dovranno quindi interagire direttamente con gli oggetti di questo package, ma utilizzeranno un'interfaccia unica per: inserire, modificare ed eliminare nodi della mappa mentale, relazioni e *percorsi*$_G$ di presentazione. La classe `ProjectModel` nasconde inoltre ai *controller*$_G$ la complessità di alcune operazioni. Ad esempio, se si intende rimuovere un nodo della *mappa mentale*$_G$ che risulta padre di un sottoalbero, è necessario eliminare in ricorsione anche i nodi figli. Questa operazione sarebbe particolarmente costosa e renderebbe molto complesso il codice del *controller*$_G$.
Nel diagramma seguente sono stati tralasciati volontariamente alcuni metodi delle classe, per mostrare in modo semplice e diretto come il pattern è stato applicato.

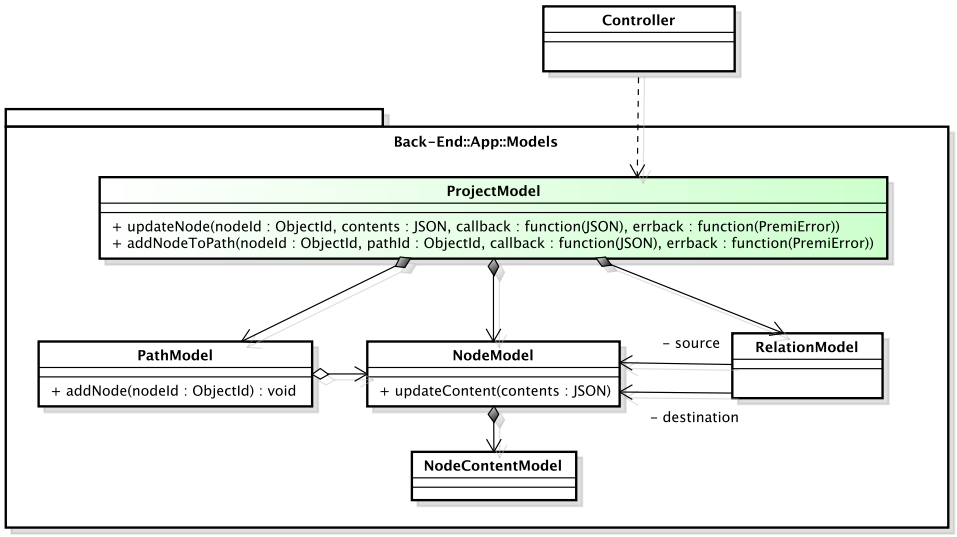


Figura 124: Facade contestualizzato

B.4 Observer

Scopo Permettere a degli oggetti di tenere sotto controllo lo stato di altri oggetti e di essere notificati quando gli oggetti osservati vengono modificati.

Motivazione In alcuni casi è necessario è necessario tenere sincronizzati vari oggetti e, nel caso un oggetto venga modificato, avere la possibilità di riflettere il cambiamento sugli altri oggetti dipendenti.
Il pattern *Observer*$_G$ permette di implementare ciò definendo due tipologie di oggetti: i *subject*, che rappresentano gli oggetti osservati, e gli observer$_G$ che rappresentano gli oggetti che osservano lo stato del `subject`. Con questo pattern è possibile quindi realizzare un sistema *publish-subscribe*, nel quale i vari *subject* si registrano per essere notificati delle modifiche subite dal *subject*.

Applicabilità Observer$_G$ si usa nei seguenti casi:

- Si vuole che il cambiamento dello stato di un oggetto provochi l'aggiornamento di altri oggetti;
- Si vuole notificare degli oggetti dell'avvenimento di un determinato evento senza sapere il loro tipo.

Utilizzo Nella parte *front-end*$_G$ dell'applicazione viene utilizzato per permettere ai vari controllers di essere notificati degli eventi sollevati dalla *libreria*$_G$ Cytoscape.js. Questi eventi infatti non vengono rilevati automaticamente da *Angular*$_G$, pertanto viene utilizzato il service `MindmapAdapterService` per creare un livello intermedio che permette di notificare alle altre classi il verificarsi degli eventi di Cytoscape. L'implementazione avviene memorizzando all'interno di `MindmapAdapterService` un'array di funzioni da invocare quando si verifica un determinato evento di Cytoscape. Il service fornisce

inoltre due metodi che permettono ai *controller*$_G$ di associare e dissociare una funzione che deve essere invocata al verificarsi di un evento. Questa particolare implementazione è possibile dal momento che le funzioni in *JavaScript*$_G$ vengono trattate come un oggetto qualsiasi. Segue un diagramma semplificato che illustra come è stato applicato il pattern.

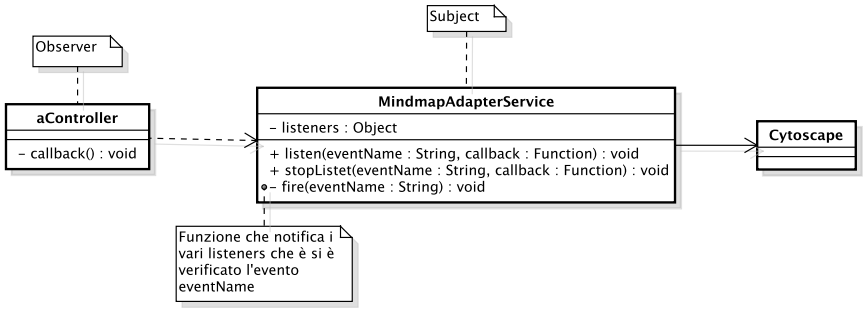


Figura 125: Observer contestualizzato

Appendice C Cytoscape

Per l'integrazione della *libreria*$_G$ Cytoscape.js con il *front-end*$_G$ dell'applicazione si è scelto di seguire quando suggerito dagli sviluppatori della *libreria*$_G$, racchiudendo cioè l'oggetto `cytoscape` all'interno di un service di *AngularJS*$_G$. Questa implementazione permette di condividere tra più componenti dell'applicazione i dati che definiscono la *mappa mentale*$_G$ e di integrare gli eventi sollevati dall'oggetto `cytoscape` con il resto dell'applicazione mediante un sistema di observer$_G$.

C.1 Organizzazione dei dati

A differenza dell'implementazione proposta si è scelto di organizzare la gestione dei dati in due array, uno per memorizzare i nodi e l'altro per le relazioni tra esse. In questo modo risulta più semplice ed efficiente tenere sincronizzati i dati visualizzati nella *view*$_G$ con quelli presenti all'interno del service.

C.2 Inizializzazione della mappa

Per rendere utilizzabile la mappa generata da Cytoscape su più *view*$_G$ e per poter ridisegnarla senza dover ricaricare tutti i dati si è scelto di separare il caricamento dei dati con la visualizzazione. Per il caricamento dei dati è presente il metodo `loadData` che si occupa di caricare i dati all'interno del service con degli oggetti definiti usando il JSON ricevuto in risposta dal *server*$_G$. Questo metodo viene invocato solamente da `ProjectService` e da `MindmapService`, così facendo viene rimosso l'onere dai *controller*$_G$ dell'inizializzazione dei dati. Per rendere visibile la *mappa mentale*$_G$ è stato definito il metodo `drawMap`, questo metodo si occupa di creare l'effettivo oggetto `cytoscape` in modo che la mappa diventi visibile all'interno del `<div>` con `id=cy`.

C.3 Eventi

Per la propagazione degli eventi generati dall'oggetto `cytoscape` si è scelto di memorizzare le funzioni di *callback*$_G$ dei vari observer$_G$ all'interno di una struttura dati associativa, associando al nome di ogni evento (chiave) un'array di funzioni da invocare quando si verifica quel determinato evento.
In questa maniera è possibile avere un livello di separazione tra gli eventi offerti da Cytoscape e gli eventi che può osservare il *controller*$_G$ in modo da definire eventi più complessi rispetto a quelli standard di Cytoscape e, nel caso di cambiamento della *libreria*$_G$, è meno probabile che sia necessario andare a modificare le classi observer$_G$. Gli eventi che sono osservabili e i relativi parametri sono:

- `node-select`:
  - `MouseEvent`: riferimento all'oggetto contenente le informazioni relative all'evento del mouse che ha causato la selezione del nodo;
  - `String`: `id` del nodo selezionato.
- `node-deselect`:
  - `MouseEvent`: riferimento all'oggetto contenente le informazioni relative all'evento del mouse che ha causato la selezione del nodo.

- `association-select`:
  - `MouseEvent`: riferimento all'oggetto contenente le informazioni relative all'evento del mouse che ha causato la selezione della relazione;
  - `String`: `id` della relazione selezionata;
  - `String`: `id` tipo della relazione selezionata.
- `association-deselect`:
  - `MouseEvent`: riferimento all'oggetto contenente le informazioni relative all'evento del mouse che ha causato la selezione del nodo.