

DeltaX

Progetto *Produlytics*

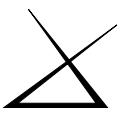
Specifiche Architetturali

Informazioni

Versione	1.0
Data Approvazione	2022-05-10
Responsabile	Giacomo Stevanato
Redattori	Leila Dardouri Alberto Lazari Christian Micheletti Riccardo Pavan Diego Stafa
Verificatori	Giacomo Stevanato Daniele Trentin Leila Dardouri Alberto Lazari Christian Micheletti Riccardo Pavan Diego Stafa Giacomo Stevanato Daniele Trentin
Destinatari	Approvato DeltaX Prof. Riccardo Cardin Prof. Tullio Vardanega Sanmarco Informatica S.p.A.
Uso	Esterno

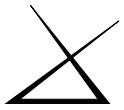
Sommario

Descrizione delle tecnologie e dell'architettura del prodotto *Produlytics*.



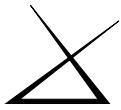
Ver.	Data	Nominativo	Ruolo	Descrizione
1.0	2022-05-10	Giacomo Stevanato	Responsabile	Approvazione del documento
0.22	2022-05-02	Daniele Trentin, Diego Staga	Amministratore, Verificatore	Fine stesura Glossario
0.21	2022-04-30	Daniele Trentin, Diego Staga	Amministratore, Verificatore	Stesura Glossario fino a J
0.20	2022-04-23	Alberto Lazati, Leila Dardouri	Progettista, Verificatrice	Progettazione di dettaglio back-end UI dashboard
0.19	2022-04-21	Riccardo Pavan, Christian Micheletti	Progettista, Verificatore	Progettazione di dettaglio front-end UI dashboard
0.18	2022-04-17	Leila Dardouri, Giacomo Stevanato	Progettista, Verificatore	Architettura delle componenti back-end UI dashboard
0.17	2022-04-14	Daniele Trentin, Christian Micheletti	Progettista, Verificatore	Architettura delle componenti front-end UI dashboard
0.16	2022-04-03	Leila Dardouri, Alberto Lazari	Progettista, Verificatore	Progettazione di dettaglio back-end UI amministrazione
0.15	2022-03-29	Riccardo Pavan, Daniele Trentin	Progettista, Verificatore	Progettazione di dettaglio front-end UI amministrazione
0.14	2022-03-26	Alberto Lazari, Leila Dardouri	Progettista, Verificatrice	Architettura delle componenti back-end UI amministrazione
0.13	2022-03-24	Riccardo Pavan, Diego Stafa	Progettista, Verificatore	Architettura delle componenti front-end UI amministrazione
0.12	2022-03-13	Diego Stafa, Riccardo Pavan	Progettista, Verificatore	Progettazione di dettaglio API rilevazioni
0.11	2022-03-10	Leila Dardouri, Alberto Lazari	Progettista, Verificatore	Progettazione di dettaglio back-end UI autenticazione
0.10	2022-03-09	Riccardo Pavan, Christian Micheletti	Progettista, Verificatore	Progettazione di dettaglio front-end UI autenticazione
0.9	2022-03-06	Giacomo Stevanato, Diego Stafa	Progettista, Verificatore	Architettura delle componenti API rilevazioni
0.8	2022-03-05	Alberto Lazari, Leila Dardouri	Progettista, Verificatrice	Architettura delle componenti back-end UI autenticazione
0.7	2022-03-05	Christian Micheletti, Daniele Trentin	Progettista, Verificatore	Architettura delle componenti front-end UI autenticazione
0.6	2022-02-27	Daniele Trentin, Alberto Lazari	Progettista, Verificatore	Stesura §3.4.4 §3.4.6

Ver.	Data	Nominativo	Ruolo	Descrizione
0.5	2022-02-27	Diego Stafa, Riccardo Pavan	Progettista, Verificatore	Configurazione back-end UI (§3.4.2)
0.4	2022-02-26	Christian Micheletti, Diego Stafa	Progettista, Verificatore	Configurazione front-end UI (§3.4.1)
0.3	2022-02-25	Giacomo Stevanato, Daniele Trentin	Progettista, Verificatore	Stesura §3.4.3 §3.4.5
0.2	2022-02-23	Alberto Lazari, Christian Micheletti	Progettista, Verificatore	Stesura da §3.1 a §3.3
0.1	2022-02-22	Riccardo Pavan, Giacomo Stevanato	Progettista, Verificatore	Stesura §1 §2
0.0	2022-02-22	Riccardo Pavan, Giacomo Stevanato	Amministratore, Verificatore	Creazione del documento

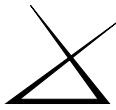


Indice

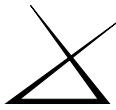
1 Introduzione	23
1.1 Scopo del documento	23
1.2 Scopo del prodotto	23
1.3 Glossario	23
1.4 Riferimenti	23
1.4.1 Normativi	23
1.4.2 Informativi	23
2 Tecnologie utilizzate	25
2.1 PostgreSQL	25
2.2 Timescale	25
2.3 Java	26
2.4 Spring Boot	26
2.5 TypeScript	26
2.6 Angular	26
2.7 D3.js	27
2.8 REST	27
2.9 JSON	27
2.10 Docker	27
2.11 Docker Compose	28
3 Architettura di sistema	29
3.1 Modello architettonico	29
3.2 Descrizione delle componenti	30
3.3 Assemblaggio delle componenti	31
3.4 Configurazione di sistema	31
3.4.1 Front-end UI	31
3.4.1.1 Definizione dell'interfaccia	31
3.4.1.1.1 Schermata di autenticazione	31
3.4.1.1.2 Barra di navigazione	31
3.4.1.1.3 Finestra di dialogo di conferma	32
3.4.1.1.4 Schermata di modifica password	32
3.4.1.1.5 Schermata di monitoraggio rilevazioni	33
3.4.1.1.6 Schermata di gestione macchine	33
3.4.1.1.7 Schermata di creazione macchina	34
3.4.1.1.8 Schermata di modifica macchina	34
3.4.1.1.9 Form per la creazione di una caratteristica	35
3.4.1.1.10 Form per la modifica di una caratteristica	35
3.4.1.1.11 Schermata di gestione utenti	36
3.4.1.1.12 Form di creazione utente	36
3.4.1.1.13 Form di modifica utente	36
3.4.1.2 Requisiti per i livelli inferiori	37
3.4.2 Back-end UI	42
3.4.2.1 Definizione dell'interfaccia	42
3.4.2.1.1 Autenticazione	42
3.4.2.1.2 Login	42
3.4.2.1.3 Logout	43
3.4.2.1.4 Modifica password	43
3.4.2.1.5 Lista macchine non archiviate	44



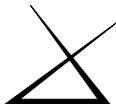
3.4.2.1.6	Lista caratteristiche non archiviate di una macchina	44
3.4.2.1.7	Rilevazioni di una caratteristica	45
3.4.2.1.8	Limiti e media di una caratteristica	46
3.4.2.1.9	Lista macchine	46
3.4.2.1.10	Modifica stato archiviazione macchina	47
3.4.2.1.11	Modifica stato attivazione macchina	47
3.4.2.1.12	Inserimento macchina	48
3.4.2.1.13	Dettagli macchina	49
3.4.2.1.14	Modifica macchina	49
3.4.2.1.15	Lista delle caratteristiche di una macchina	50
3.4.2.1.16	Modifica stato archiviazione caratteristica	50
3.4.2.1.17	Inserimento caratteristica	51
3.4.2.1.18	Modifica caratteristica	52
3.4.2.1.19	Lista utenti	53
3.4.2.1.20	Modifica stato archiviazione utente	53
3.4.2.1.21	Inserimento utente	54
3.4.2.1.22	Modifica utente	54
3.4.2.2	Requisiti per i livelli inferiori	55
3.4.3	API rilevazioni	57
3.4.3.1	Definizione dell'interfaccia	57
3.4.3.1.1	Rilevazioni	57
3.4.3.2	Requisiti per i livelli inferiori	57
3.4.4	Database	58
3.4.4.1	Definizione dell'interfaccia	58
3.4.5	Libreria di persistenza	61
3.4.6	Diagrammi di sequenza	65
3.4.6.1	Elenco delle macchine	65
3.4.6.2	Invio delle rilevazioni	66
4	Architettura delle componenti	68
4.1	Front-end UI	68
4.1.1	HttpInterceptor	69
4.1.1.1	HttpInterceptorService	70
4.1.2	Guards	70
4.1.2.1	AdminGuard	71
4.1.2.2	LoginGuard	71
4.1.2.3	AuthenticatedUserGuard	71
4.1.3	Schermata di autenticazione	72
4.1.3.1	LoginComponent	72
4.1.3.2	LoginAbstractService	73
4.1.3.3	LoginService	74
4.1.4	Barra di navigazione	74
4.1.4.1	ToolbarComponent	75
4.1.5	Schermata di modifica password	76
4.1.5.1	ModifyPwComponent	76
4.1.5.2	ModifyPwAbstractService	77
4.1.5.3	ModifyPwService	77
4.1.6	Schermata di monitoraggio rilevazioni	78
4.1.6.1	SelectionComponent	78
4.1.6.2	SelectionDataSource	79
4.1.6.3	DeviceSelection	80
4.1.6.4	DeviceNode	80



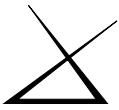
4.1.6.5	CharacteristicNode	80
4.1.6.6	ChartComponent	81
4.1.6.7	ChartAbstractService	83
4.1.6.8	ChartService	83
4.1.6.9	DatePickerDialogComponent	83
4.1.7	Schermata di gestione macchine	84
4.1.7.1	DevicesComponent	85
4.1.7.2	DevicesModule	86
4.1.7.3	DeviceDataSource	86
4.1.7.4	DeviceAbstractService	87
4.1.7.5	DeviceService	87
4.1.7.6	Device	88
4.1.8	Schermata di creazione macchina	88
4.1.8.1	NewDeviceComponent	89
4.1.8.2	NewDeviceAbstractService	90
4.1.8.3	NewDeviceService	90
4.1.9	Schermata di modifica macchina	90
4.1.9.1	DeviceDetail	90
4.1.9.2	CharacteristicDataSource	92
4.1.9.3	DeviceDetailResolver	93
4.1.9.4	CharacteristicAbstractService	93
4.1.9.5	CharacteristicService	94
4.1.9.6	Characteristic	94
4.1.9.7	CharacteristicCreationCommand	94
4.1.9.8	FindDeviceAbstractService	94
4.1.9.9	FindDeviceService	95
4.1.9.10	UpdateDeviceAbstractService	95
4.1.9.11	UpdateDeviceService	95
4.1.10	Form per la creazione di una caratteristica	96
4.1.10.1	NewCharacteristicDialogComponent	96
4.1.10.2	CharacteristicFormComponent	97
4.1.11	Form per la modifica di una caratteristica	98
4.1.11.1	UpdateCharacteristicDialogComponent	98
4.1.11.2	UpdateCharacteristicAbstractService	99
4.1.11.3	UpdateCharacteristicService	99
4.1.11.4	CharacteristicUpdateCommand	100
4.1.12	Schermata di gestione utenti	100
4.1.12.1	AccountsComponent	100
4.1.12.2	AccountsModule	101
4.1.12.3	AccountsDataSource	102
4.1.12.4	AccountAbstractService	102
4.1.12.5	AccountService	102
4.1.12.6	Account	103
4.1.13	Form di creazione e modifica utente	103
4.1.13.1	SaveAccountAbstractService	104
4.1.13.2	SaveAccountService	105
4.1.13.3	AccountSaveCommand	105
4.1.14	Altre componenti	105
4.1.14.1	ConfirmDialogComponent	105
4.1.14.2	ErrorDialogComponent	106
4.1.15	Classi di utilità	106
4.1.15.1	NotificationService	106



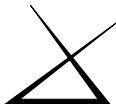
4.2 Back-end UI	108
4.2.1 Controller	109
4.2.1.1 AccountsController	109
4.2.1.2 AdminsAccountsController	109
4.2.1.3 AdminsDevicesController	110
4.2.1.4 DevicesController	112
4.2.1.5 AdminsCharacteristicsController	112
4.2.1.6 CharacteristicsController	113
4.2.1.7 DetectionsController	114
4.2.1.8 LoginController	115
4.2.2 Gestione delle eccezioni	116
4.2.2.1 BusinessExceptionHandler	116
4.2.2.2 BusinessException	117
4.2.2.3 ErrorType	117
4.2.3 Login	118
4.2.3.1 CustomAccountDetails	118
4.2.3.2 Account	119
4.2.3.3 ProdulyticsGrantedAuthority	119
4.2.3.4 UserDetailsAdapter	119
4.2.3.5 FindAccountUseCase	120
4.2.3.6 BusinessException	120
4.2.3.7 FindAccountService	120
4.2.3.8 FindAccountPort	120
4.2.3.9 AccountAdapter	121
4.2.3.10 AccountRepository	121
4.2.3.11 SecurityConfiguration	122
4.2.3.12 EncoderConfig	122
4.2.3.13 LoginController	122
4.2.4 Modifica password	123
4.2.4.1 AccountController	123
4.2.4.2 UpdateAccountPasswordUseCase	123
4.2.4.3 AccountPasswordToUpdate	123
4.2.4.4 BusinessException	124
4.2.4.5 UpdateAccountPasswordService	124
4.2.4.6 PasswordMatcherPort	124
4.2.4.7 PasswordMatcherAdapter	125
4.2.4.8 PasswordEncoderPort	125
4.2.4.9 PasswordEncoderAdapter	125
4.2.4.10 EncoderConfig	126
4.2.4.11 UpdateAccountPasswordPort	126
4.2.4.12 FindAccountPort	126
4.2.4.13 Account	126
4.2.4.14 AccountAdapter	126
4.2.4.15 AccountRepository	126
4.2.5 Lista macchine non archiviate	127
4.2.5.1 DevicesController	127
4.2.5.2 GetUnarchivedDevicesUseCase	127
4.2.5.3 TinyDevice	127
4.2.5.4 GetUnarchivedDevicesService	127
4.2.5.5 GetUnarchivedDevicesPort	128
4.2.5.6 UnarchivedDeviceAdapter	128
4.2.5.7 DeviceRepository	128



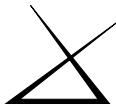
4.2.6	Lista caratteristiche non archiviate di una macchina	129
4.2.6.1	CharacteristicsController	129
4.2.6.2	GetUnarchivedCharacteristicsUseCase	129
4.2.6.3	BusinessException	129
4.2.6.4	TinyCharacteristic	130
4.2.6.5	GetUnarchivedCharacteristicsService	130
4.2.6.6	GetUnarchivedDevicesPort	130
4.2.6.7	FindAllUnarchivedCharacteristicsPort	130
4.2.6.8	UnarchivedCharacteristicAdapter	131
4.2.6.9	CharacteristicRepository	131
4.2.7	Rilevazioni di una caratteristica	132
4.2.7.1	DetectionsController	132
4.2.7.2	DetectionFilters	132
4.2.7.3	GetDetectionsUseCase	133
4.2.7.4	DetectionsGroup	133
4.2.7.5	Detection	133
4.2.7.6	BusinessException	133
4.2.7.7	GetDetectionsService	133
4.2.7.8	FindAllDetectionsPort	134
4.2.7.9	FindCharacteristicLimitsPort	134
4.2.7.10	DetectionAdapter	134
4.2.7.11	DetectionsRepository	135
4.2.8	Limits e media di una caratteristica	136
4.2.8.1	CharacteristicsController	136
4.2.8.2	GetLimitsUseCase	136
4.2.8.3	BusinessException	136
4.2.8.4	CharacteristicLimits	137
4.2.8.5	GetLimitsService	137
4.2.8.6	GetUnarchivedDevicesPort	137
4.2.8.7	FindCharacteristicLimitsPort	137
4.2.8.8	UnarchivedCharacteristicAdapter	138
4.2.8.9	CharacteristicRepository	138
4.2.9	Lista macchine	138
4.2.9.1	AdminsDevicesController	138
4.2.9.2	GetDevicesUseCase	138
4.2.9.3	Device	138
4.2.9.4	GetDevicesService	139
4.2.9.5	GetDevicesPort	139
4.2.9.6	AdminDeviceAdapter	139
4.2.9.7	DeviceRepository	140
4.2.10	Archiviazione macchina	141
4.2.10.1	AdminsDevicesController	141
4.2.10.2	UpdateDeviceArchiveStatusUseCase	141
4.2.10.3	UpdateDeviceArchiveStatusService	141
4.2.10.4	DeviceArchiveStatus	142
4.2.10.5	FindDetailedDevicePort	142
4.2.10.6	DetailedDevice	142
4.2.10.7	UpdateDeviceArchiveStatusPort	142
4.2.10.8	AdminDeviceAdapter	143
4.2.10.9	DeviceRepository	143
4.2.11	Disattivazione macchina	143
4.2.11.1	AdminsDevicesController	143



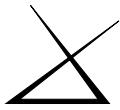
4.2.11.2	UpdateDeviceDeactivateStatusUseCase	143
4.2.11.3	DeviceDeactivateStatus	143
4.2.11.4	BusinessException	144
4.2.11.5	UpdateDeviceDeactivateStatusService	144
4.2.11.6	FindDetailedDevicePort	144
4.2.11.7	UpdateDeviceDeactivateStatusPort	144
4.2.11.8	AdminDeviceAdapter	144
4.2.11.9	DeviceRepository	144
4.2.12	Dettagli macchina	145
4.2.12.1	AdminsDevicesController	145
4.2.12.2	GetDeviceDetailsUseCase	145
4.2.12.3	DetailedDevice	145
4.2.12.4	GetDeviceDetailsService	145
4.2.12.5	FindDetailedDevicePort	146
4.2.12.6	AdminDeviceAdapter	146
4.2.12.7	DeviceRepository	146
4.2.13	Modifica macchina	146
4.2.13.1	AdminsDevicesController	146
4.2.13.2	UpdateDeviceNameUseCase	146
4.2.13.3	TinyDevice	147
4.2.13.4	BusinessException	147
4.2.13.5	UpdateDeviceNameService	147
4.2.13.6	FindDetailedDevicePort	147
4.2.13.7	DetailedDevice	147
4.2.13.8	FindTinyDeviceByNamePort	147
4.2.13.9	UpdateDeviceNamePort	148
4.2.13.10	AdminDeviceAdapter	148
4.2.13.11	DeviceRepository	148
4.2.14	Inserimento macchina	148
4.2.14.1	AdminsDevicesController	148
4.2.14.2	InsertDeviceUseCase	149
4.2.14.3	DeviceToInsert	149
4.2.14.4	NewCharacteristic	149
4.2.14.5	InsertDeviceService	149
4.2.14.6	CreateDevice	150
4.2.14.7	NewDevice	150
4.2.14.8	FindTinyDeviceByNamePort	151
4.2.14.9	TinyDevice	151
4.2.14.10	InsertDevicePort	151
4.2.14.11	AdminDeviceAdapter	151
4.2.14.12	DeviceRepository	151
4.2.14.13	InsertCharacteristicUseCase	151
4.2.15	Lista delle caratteristiche di una macchina	152
4.2.15.1	AdminsCharacteristicsController	152
4.2.15.2	GetCharacteristicsUseCase	152
4.2.15.3	BusinessException	152
4.2.15.4	Characteristic	152
4.2.15.5	GetCharacteristicsService	153
4.2.15.6	FindAllCharacteristicsPort	153
4.2.15.7	AdminCharacteristicAdapter	153
4.2.15.8	CharacteristicRepository	154
4.2.16	Archiviazione caratteristica	155



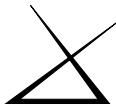
4.2.16.1	AdminsCharacteristicsController	155
4.2.16.2	UpdateCharacteristicArchiveStatusUseCase	155
4.2.16.3	CharacteristicArchiveStatus	155
4.2.16.4	UpdateCharacteristicArchiveStatusService	156
4.2.16.5	FindDetailedCharacteristicPort	156
4.2.16.6	ConvertCharacteristic	157
4.2.16.7	UpdateCharacteristicPort	157
4.2.16.8	AdminCharacteristicAdapter	157
4.2.16.9	CharacteristicRepository	157
4.2.17	Inserimento caratteristica	158
4.2.17.1	AdminsCharacteristicsController	158
4.2.17.2	InsertCharacteristicUseCase	158
4.2.17.3	NewCharacteristic	158
4.2.17.4	InsertCharacteristicService	158
4.2.17.5	CharacteristicConstraints	159
4.2.17.6	InsertCharacteristicPort	159
4.2.17.7	FindDetailedDevicePort	159
4.2.17.8	FindDetailedCharacteristicPort	159
4.2.17.9	ConvertCharacteristic	159
4.2.17.10	AdminCharacteristicAdapter	160
4.2.17.11	CharacteristicRepository	160
4.2.18	Modifica caratteristica	160
4.2.18.1	AdminsCharacteristicsController	160
4.2.18.2	UpdateCharacteristicUseCase	160
4.2.18.3	CharacteristicToUpdate	161
4.2.18.4	BusinessException	161
4.2.18.5	UpdateCharacteristicService	161
4.2.18.6	CharacteristicConstraints	162
4.2.18.7	FindDetailedCharacteristicPort	162
4.2.18.8	ConvertCharacteristic	162
4.2.18.9	UpdateCharacteristicPort	162
4.2.18.10	AdminCharacteristicAdapter	162
4.2.18.11	CharacteristicRepository	162
4.2.19	Lista utenti	163
4.2.19.1	AdminsAccountsController	163
4.2.19.2	GetTinyAccountsUseCase	163
4.2.19.3	TinyAccount	163
4.2.19.4	GetTinyAccountsService	163
4.2.19.5	GetTinyAccountsPort	164
4.2.19.6	AdminAccountAdapter	164
4.2.19.7	AccountRepository	165
4.2.20	Archiviazione utente	165
4.2.20.1	AdminsAccountsController	165
4.2.20.2	UpdateAccountArchiveStatusUseCase	165
4.2.20.3	AccountArchiveStatus	166
4.2.20.4	BusinessException	166
4.2.20.5	UpdateAccountArchiveStatusService	166
4.2.20.6	Account	166
4.2.20.7	FindAccountByAdminPort	166
4.2.20.8	UpdateAccountArchiveStatusPort	167
4.2.20.9	AdminAccountAdapter	167
4.2.20.10	AccountRepository	167



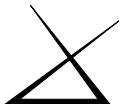
4.2.21	Inserimento utente	167
4.2.21.1	AdminsAccountsController	168
4.2.21.2	InsertAccountUseCase	168
4.2.21.3	AccountToInsert	168
4.2.21.4	BusinessException	168
4.2.21.5	InsertAccountService	168
4.2.21.6	FindAccountByAdminPort	169
4.2.21.7	Account	169
4.2.21.8	InsertAccountPort	169
4.2.21.9	AdminAccountAdapter	169
4.2.21.10	AccountRepository	169
4.2.22	Modifica utente	170
4.2.22.1	AdminsAccountsController	170
4.2.22.2	UpdateAccountByAdminUseCase	170
4.2.22.3	AccountUpdatedByAdmin	170
4.2.22.4	BusinessException	171
4.2.22.5	UpdateAccountByAdminService	171
4.2.22.6	PasswordEncoderPort	171
4.2.22.7	FindAccountByAdminPort	171
4.2.22.8	Account	171
4.2.22.9	UpdateAccountByAdminPort	171
4.2.22.10	AdminAccountAdapter	172
4.2.22.11	AccountRepository	172
4.2.23	Configuration	172
4.2.23.1	AccountsConfiguration	172
4.2.23.2	AdminsAccountsConfiguration	173
4.2.23.3	AdminsDevicesConfiguration	175
4.2.23.4	DetectionsConfiguration	178
4.2.23.5	DevicesConfiguration	179
4.3	API rilevazioni	181
4.3.1	Controller	182
4.3.1.1	DetectionsController	182
4.3.1.2	ProcessIncomingDetectionUseCase	183
4.3.1.3	IncomingDetection	183
4.3.1.4	BusinessException	183
4.3.2	Gestione delle eccezioni	184
4.3.2.1	BusinessExceptionHandler	184
4.3.2.2	BusinessException	184
4.3.2.3	ErrorType	185
4.3.3	Adapter	185
4.3.3.1	DetectionsAdapter	186
4.3.3.2	DetectionRepository	187
4.3.3.3	DeviceRepository	188
4.3.3.4	CharacteristicRepository	188
4.3.3.5	LimitsEntity	189
4.3.3.6	FindCharacteristicByNamePort	189
4.3.3.7	FindDeviceByApiKeyPort	190
4.3.3.8	FindLimitsPort	190
4.3.3.9	InsertDetectionPort	190
4.3.3.10	FindLastDetectionsPort	191
4.3.3.11	MarkOutlierPort	191
4.3.4	Service	192



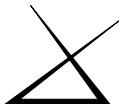
4.3.4.1	DetectionsService	192
4.3.4.2	DetectionValidator	193
4.3.4.3	DetectionQueue	193
4.3.4.4	Detection	193
4.3.4.5	CharacteristicId	193
4.3.4.6	ProcessIncomingDetectionUseCase	194
4.3.4.7	BusinessException	194
4.3.5	Validator	194
4.3.5.1	DetectionValidatorImpl	194
4.3.5.2	DeviceInfo	195
4.3.5.3	CharacteristicInfo	195
4.3.5.4	DetectionValidator	195
4.3.5.5	FindDeviceByApiKeyPort	195
4.3.5.6	FindCharacteristicByNamePort	195
4.3.6	Queue	196
4.3.6.1	DetectionQueueImpl	196
4.3.6.2	DetectionSeriesFactory	197
4.3.6.3	DetectionSeries	197
4.3.6.4	DetectionSeriesImplFactory	197
4.3.6.5	DetectionQueue	198
4.3.6.6	DetectionSeriesImpl	198
4.3.7	Series	198
4.3.7.1	DetectionSeriesImpl	199
4.3.7.2	ControlLimitsCalculator	199
4.3.7.3	SeriesPortFacade	199
4.3.7.4	ControlChartsGroup	200
4.3.7.5	MarkableDetection	200
4.3.7.6	MarkableDetectionAdapter	200
4.3.7.7	ControlLimits	201
4.3.8	SeriesPortFacade	202
4.3.8.1	SeriesPortFacadeImpl	202
4.3.8.2	SeriesPortFacade	203
4.3.8.3	InsertDetectionPort	203
4.3.8.4	FindLastDetectionsPort	203
4.3.8.5	MarkOutlierPort	203
4.3.9	ControlLimitsCalculator	203
4.3.9.1	ControlLimitsCalculatorImpl	204
4.3.9.2	LimitsInfo	204
4.3.9.3	TechnicalLimits	204
4.3.9.4	MeanStddev	204
4.3.9.5	ControlLimitsCalculator	204
4.3.9.6	ControlLimits	205
4.3.9.7	FindLimitsPort	205
4.3.10	ControlChartsGroup	205
4.3.10.1	ControlChartsGroupImpl	205
4.3.10.2	ControlChart	206
4.3.10.3	ControlChartBeyondLimits	206
4.3.10.4	ControlChartMixture	207
4.3.10.5	ControlChartOverControl	207
4.3.10.6	ControlChartStratification	207
4.3.10.7	ControlChartTrend	208
4.3.10.8	ControlChartZoneA	208



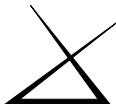
4.3.10.9	ControlChartZoneB	208
4.3.10.10	ControlChartZoneC	209
4.3.10.11	ControlChartUtils	209
4.3.10.12	ControlChartsGroup	209
4.3.11	Configuration	210
4.3.11.1	DetectionsConfiguration	210
4.3.11.2	DetectionValidatorImpl	212
4.3.11.3	ControlChartsGroupImpl	212
4.3.11.4	SeriesPortFacadeImpl	212
4.3.11.5	ControlLimitsCalculatorImpl	212
4.3.11.6	DetectionSeriesImplFactory	212
4.3.11.7	DetectionQueueImpl	212
4.3.11.8	DetectionsService	212
5	Progettazione di dettaglio	213
5.1	Front-end UI	213
5.1.1	LoginComponent	213
5.1.1.1	ngOnInit	213
5.1.1.2	onSubmit	213
5.1.2	ToolbarComponent	213
5.1.2.1	openPwDialog	213
5.1.2.2	isLoggedIn	214
5.1.2.3	isAdmin	214
5.1.2.4	getUsername	214
5.1.2.5	logout	214
5.1.3	ModifyPwComponent	214
5.1.3.1	checkPasswords	215
5.1.3.2	confirm	215
5.1.3.3	cancel	215
5.1.4	SelectionComponent	215
5.1.4.1	hasChildren	215
5.1.4.2	nodeIsChecked	215
5.1.4.3	toggleNodeCheck	216
5.1.4.4	notifyChange	216
5.1.5	SelectionDataSource	216
5.1.5.1	connect	216
5.1.5.2	disconnect	216
5.1.5.3	isNodeLoading	216
5.1.5.4	handleChange	216
5.1.5.5	addCharacteristics	216
5.1.5.6	removeCharacteristics	216
5.1.5.7	getChildrenForNode	217
5.1.6	ChartComponent	217
5.1.6.1	ngAfterViewInit	218
5.1.6.2	ngOnDestroy	218
5.1.6.3	getChartWidth	218
5.1.6.4	getChartHeight	218
5.1.6.5	getData	218
5.1.6.6	createChart	218
5.1.6.7	drawChart	219
5.1.6.8	subscribeToUpdates	219
5.1.6.9	clearChart	219



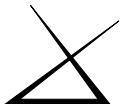
5.1.6.10 refresh	219
5.1.7 DatePickerDialogComponent	219
5.1.7.1 confirm	220
5.1.7.2 cancel	220
5.1.8 DevicesComponent	220
5.1.8.1 ngOnInit	221
5.1.8.2 createDevice	221
5.1.8.3 openDeviceDetail	221
5.1.8.4 toggleActivationDevice	221
5.1.8.5 toggleStatusDevice	221
5.1.8.6 private initTable	221
5.1.9 DeviceDataSource	221
5.1.9.1 connect	221
5.1.9.2 disconnect	221
5.1.9.3 setData	222
5.1.10 NewDeviceComponent	222
5.1.10.1 ngOnInit	223
5.1.10.2 openNewCharacteristicDialog	223
5.1.10.3 insertDevice	223
5.1.10.4 private initForm	223
5.1.11 DeviceDetailComponent	223
5.1.11.1 ngOnInit	224
5.1.11.2 updateDeviceName	224
5.1.11.3 openNewCharacteristicDialog	224
5.1.11.4 openUpdateCharacteristicFormDialog	224
5.1.11.5 notifyCopy	224
5.1.11.6 toggleCharacteristicStatus	224
5.1.11.7 private initTable	225
5.1.12 DeviceDetailResolver	225
5.1.12.1 resolve	225
5.1.13 CharacteristicDataSource	225
5.1.13.1 get data	225
5.1.13.2 set data	225
5.1.13.3 connect	225
5.1.13.4 disconnect	225
5.1.14 NewCharacteristicDialogComponent	225
5.1.14.1 close	225
5.1.14.2 confirm	225
5.1.15 UpdateCharacteristicDialogComponent	226
5.1.15.1 close	226
5.1.15.2 confirm	226
5.1.16 CharacteristicFormComponent	226
5.1.16.1 ngOnInit	226
5.1.16.2 requireData	227
5.1.17 AccountsComponent	227
5.1.17.1 ngOnInit	227
5.1.17.2 openNewAccountDialog	228
5.1.17.3 openEditAccountDialog	228
5.1.17.4 toggleAccountStatus	228
5.1.17.5 isLoggedInUser	228
5.1.17.6 private initTable	228
5.1.18 AccountsDataSource	228



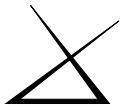
5.1.18.1	connect	228
5.1.18.2	disconnect	228
5.1.18.3	setData	228
5.1.18.4	sortData	229
5.1.19	AccountFormDialogComponent	229
5.1.19.1	ngOnInit	230
5.1.19.2	cancel	230
5.1.19.3	confirm	230
5.1.19.4	toggleChangePassword	230
5.1.19.5	private insertAccount	230
5.1.19.6	private updateAccount	230
5.1.19.7	private showError	230
5.2	Back-end UI	231
5.2.1	AccountsController	231
5.2.1.1	updateAccountPassword	231
5.2.2	AdminsAccountsController	231
5.2.2.1	insertAccount	231
5.2.2.2	getAccounts	231
5.2.2.3	updateAccount	231
5.2.2.4	updateAccountArchiveStatus	231
5.2.3	AdminsCharacteristicsController	231
5.2.3.1	insertCharacteristic	231
5.2.3.2	getCharacteristics	231
5.2.3.3	updateCharacteristicArchiveStatus	232
5.2.3.4	updateCharacteristic	232
5.2.4	AdminsDevicesController	232
5.2.4.1	insertDevice	232
5.2.4.2	getDevices	232
5.2.4.3	getDeviceDetails	232
5.2.4.4	updateDeviceName	232
5.2.4.5	updateDeviceArchiveStatus	232
5.2.4.6	updateDeviceDeactivateStatus	233
5.2.5	CharacteristicsController	233
5.2.5.1	getUnarchivedCharacteristics	233
5.2.5.2	getCharacteristicLimits	233
5.2.6	DetectionsController	233
5.2.6.1	getCharacteristicDetections	233
5.2.7	DevicesController	233
5.2.7.1	getUnarchivedDevices	233
5.2.8	LoginController	233
5.2.8.1	login	233
5.2.9	BusinessExceptionHandler	234
5.2.9.1	handleStatusException	234
5.2.10	BusinessException	234
5.2.10.1	getCode	234
5.2.10.2	getType	234
5.2.11	AccountAdapter	234
5.2.11.1	findByUsername	234
5.2.11.2	updateAccountPassword	234
5.2.12	AdminAccountAdapter	234
5.2.12.1	updateAccountArchiveStatus	234
5.2.12.2	findByUsername	234



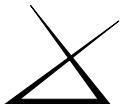
5.2.12.3	updateAccount	234
5.2.12.4	insertAccount	235
5.2.12.5	getTinyAccounts	235
5.2.13	AdminDeviceAdapter	235
5.2.13.1	getDevices	235
5.2.13.2	findTinyDevice	235
5.2.13.3	findDetailedDevice	235
5.2.13.4	updateDeviceArchiveStatus	235
5.2.13.5	updateDeviceDeactivateStatus	235
5.2.13.6	updateDeviceName	235
5.2.13.7	insertDevice	235
5.2.14	AdminCharacteristicAdapter	235
5.2.14.1	findAllByDeviceId	235
5.2.14.2	findByCharacteristic	236
5.2.14.3	findByDeviceAndName	236
5.2.14.4	insertByDevice	236
5.2.14.5	updateCharacteristic	236
5.2.15	DetectionAdapter	236
5.2.15.1	findAllByCharacteristic	236
5.2.16	PasswordEncoderAdapter	236
5.2.16.1	encode	236
5.2.17	PasswordMatcherAdapter	236
5.2.17.1	matches	236
5.2.18	UnarchivedCharacteristicAdapter	236
5.2.18.1	findAllByDeviceId	236
5.2.18.2	findByCharacteristic	237
5.2.19	UserDetailsAdapter	237
5.2.19.1	loadUserByUsername	237
5.2.20	DetectionRepository	237
5.2.20.1	findByCharacteristicAndCreationTimeGreaterThanOrEqual	237
5.2.21	FindAccountService	237
5.2.21.1	findByUsername	237
5.2.22	GetTinyAccountsService	237
5.2.22.1	getTinyAccounts	237
5.2.23	InsertAccountService	237
5.2.23.1	insertAccount	237
5.2.24	UpdateAccountArchiveStatusService	238
5.2.24.1	updateAccountArchiveStatus	238
5.2.25	UpdateAccountByAdminService	238
5.2.25.1	updateByUsername	238
5.2.26	UpdateAccountPasswordService	238
5.2.26.1	updatePasswordByUsername	238
5.2.27	CreateDevice	239
5.2.27.1	generateApiKey	239
5.2.27.2	createDevice	239
5.2.28	GetDeviceDetailsService	239
5.2.28.1	getDeviceDetails	239
5.2.29	GetDevicesService	239
5.2.29.1	getDevices	239
5.2.30	GetUnarchivedDevicesService	239
5.2.30.1	getUnarchivedDevices	239
5.2.31	InsertDeviceService	239



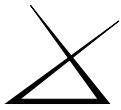
5.2.31.1 insertDevice	239
5.2.32 UpdateDeviceArchiveStatusService	240
5.2.32.1 updateDeviceArchiveStatus	240
5.2.33 UpdateDeviceDeactivateStatusService	240
5.2.33.1 updateDeviceDeactivateStatus	240
5.2.34 UpdateDeviceNameService	240
5.2.34.1 updateDeviceName	240
5.2.35 CharacteristicConstraints	240
5.2.35.1 characteristicConstraintsOk	240
5.2.36 ConvertCharacteristic	240
5.2.36.1 toDetailed	240
5.2.36.2 toEntity	240
5.2.37 GetCharacteristicsService	241
5.2.37.1 getByDevice	241
5.2.38 InsertCharacteristicService	241
5.2.38.1 insertByDevice	241
5.2.39 UpdateCharacteristicArchiveStatusService	241
5.2.40 UpdateCharacteristicService	241
5.2.41 GetUnarchivedCharacteristicsService	242
5.2.41.1 getByDevice	242
5.2.42 GetDetectionsService	242
5.2.42.1 listByCharacteristic	242
5.2.43 GetLimitsService	243
5.2.43.1 getByCharacteristic	243
5.2.44 CustomAccountDetails	243
5.2.44.1 getUsername	243
5.2.44.2 getPassword	243
5.2.44.3 getAuthorities	243
5.2.44.4 isAccountNonExpired	243
5.2.44.5 isAccountNonLocked	243
5.2.44.6 isCredentialsNonExpired	243
5.2.44.7 isEnabled	243
5.2.45 ProdulyticsGrantedAuthority	244
5.2.45.1 getAuthority	244
5.2.46 AccountsConfiguration	244
5.2.46.1 findAccountUseCase	244
5.2.46.2 updateAccountPasswordUseCase	244
5.2.47 AdminsAccountsConfiguration	244
5.2.47.1 getTinyAccountsUseCase	244
5.2.47.2 insertAccountUseCase	244
5.2.47.3 updateAccountArchiveStatusUseCase	244
5.2.47.4 updateAccountByAdminUseCase	244
5.2.48 AdminsDevicesConfiguration	244
5.2.48.1 getCharacteristicsUseCase	244
5.2.48.2 getDeviceDetailsUseCase	244
5.2.48.3 getDevicesUseCase	244
5.2.48.4 insertCharacteristicUseCase	244
5.2.48.5 insertDeviceUseCase	245
5.2.48.6 updateCharacteristicArchiveStatusUseCase	245
5.2.48.7 updateCharacteristicUseCase	245
5.2.48.8 updateDeviceArchiveStatusUseCase	245
5.2.48.9 updateDeviceDeactivateStatusUseCase	245



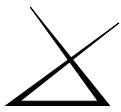
5.2.48.10 updateDeviceNameUseCase	245
5.2.49 DetectionsConfiguration	245
5.2.49.1 getDetectionsUseCase	245
5.2.50 DevicesConfiguration	245
5.2.50.1 getLimitsUseCase	245
5.2.50.2 getUnarchivedCharacteristicsUseCase	245
5.2.50.3 getUnarchivedDevices	245
5.2.51 EncoderConfig	245
5.2.51.1 getEncoder	245
5.2.52 SecurityConfiguration	246
5.2.52.1 authenticationProvider	246
5.2.52.2 configure	246
5.2.53 UserDetailsAdapterConfiguration	246
5.2.53.1 userDetailsService	246
5.2.54 Diagrammi di sequenza	246
5.2.54.1 Modifica password	247
5.2.54.2 Lista macchine non archiviate	247
5.2.54.3 Lista caratteristiche non archiviate di una macchina	248
5.2.54.4 Rilevazioni di una caratteristica	248
5.2.54.5 Limiti e media di una caratteristica	249
5.2.54.6 Lista macchine	250
5.2.54.7 Archiviazione macchina	250
5.2.54.8 Disattivazione macchina	251
5.2.54.9 Dettagli macchina	251
5.2.54.10 Modifica macchina	252
5.2.54.11 Inserimento macchina	252
5.2.54.12 Lista delle caratteristiche di una macchina	253
5.2.54.13 Archiviazione caratteristica	254
5.2.54.14 Inserimento caratteristica	254
5.2.54.15 Modifica caratteristica	255
5.2.54.16 Lista utenti	256
5.2.54.17 Archiviazione utente	256
5.2.54.18 Inserimento utente	257
5.2.54.19 Modifica utente	257
5.2.54.20 Gestione di un errore	258
5.3 API rilevazioni	258
5.3.1 DetectionsController	258
5.3.1.1 addDetection	258
5.3.2 BusinessExceptionHandler	258
5.3.2.1 handleStatusException	258
5.3.3 BusinessException	259
5.3.3.1 getCode	259
5.3.3.2 getType	259
5.3.4 DetectionsAdapter	259
5.3.4.1 findDeviceByApiKey	259
5.3.4.2 findCharacteristicByName	259
5.3.4.3 findLastDetections	259
5.3.4.4 insertDetection	259
5.3.4.5 findLimits	259
5.3.4.6 markOutlier	259
5.3.5 DetectionRepository	260
5.3.5.1 findLastDetectionsById	260



5.3.5.2	markOutlier	260
5.3.6	CharacteristicRepository	260
5.3.6.1	findLimits	260
5.3.7	DetectionValidatorImpl	260
5.3.7.1	validateAndFindId	260
5.3.8	DetectionQueueImpl	261
5.3.8.1	DetectionQueueImpl	261
5.3.8.2	enqueueDetection	261
5.3.8.3	close	261
5.3.8.4	handleDetectionGroup	261
5.3.9	DetectionSeriesImplFactory	261
5.3.9.1	createSeries	261
5.3.10	DetectionSeriesImpl	262
5.3.10.1	insertDetection	262
5.3.10.2	detectionsForControlChart	262
5.3.11	MarkableDetectionAdapter	262
5.3.11.1	value	262
5.3.11.2	markOutlier	262
5.3.12	ControlLimits	262
5.3.12.1	mean	262
5.3.12.2	lowerBCLimit	262
5.3.12.3	upperBCLimit	262
5.3.12.4	lowerABLlimit	263
5.3.12.5	upperABLlimit	263
5.3.12.6	stddev	263
5.3.13	SeriesPortFacadeImpl	263
5.3.13.1	insertDetection	263
5.3.13.2	findLastDetections	263
5.3.13.3	markOutlier	263
5.3.14	ControlLimitsCalculatorImpl	263
5.3.14.1	calculateControlLimits	263
5.3.14.2	fromMeanStddev	263
5.3.14.3	fromTechnicalLimits	264
5.3.15	ControlChartsGroupImpl	264
5.3.15.1	requiredDetectionCount	264
5.3.15.2	analyzeDetections	264
5.3.15.3	cutLastDetections	264
5.3.16	ControlChartBeyondLimits	264
5.3.16.1	requiredDetectionCount	264
5.3.16.2	analyzeDetections	264
5.3.17	ControlChartMixture	264
5.3.17.1	requiredDetectionCount	264
5.3.17.2	analyzeDetections	265
5.3.18	ControlChartOverControl	265
5.3.18.1	requiredDetectionCount	265
5.3.18.2	analyzeDetections	265
5.3.19	ControlChartStratification	265
5.3.19.1	requiredDetectionCount	265
5.3.19.2	analyzeDetections	265
5.3.20	ControlChartTrend	265
5.3.20.1	requiredDetectionCount	265
5.3.20.2	analyzeDetections	265

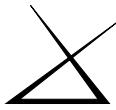


5.3.21	ControlChartZoneA	265
5.3.21.1	requiredDetectionCount	265
5.3.21.2	analyzeDetections	265
5.3.22	ControlChartZoneB	266
5.3.22.1	requiredDetectionCount	266
5.3.22.2	analyzeDetections	266
5.3.23	ControlChartZoneC	266
5.3.23.1	requiredDetectionCount	266
5.3.23.2	analyzeDetections	266
5.3.24	ControlChartUtils	266
5.3.24.1	windows	266
5.3.24.2	markAll	266
5.3.25	DetectionsConfiguration	267
5.3.25.1	createDetectionValidator	267
5.3.25.2	createControlCharts	267
5.3.25.3	createSeriesFacadePort	267
5.3.25.4	controlLimitsCalculator	267
5.3.25.5	createDetectionSeriesFactory	267
5.3.25.6	createDetectionQueue	267
5.3.25.7	createProcessDetectionUseCase	267
5.3.26	Diagrammi di sequenza	267
5.3.26.1	Ricezione di una rilevazione	267
5.3.26.2	Gestione di un errore	268
5.3.26.3	Analisi di una rilevazione	269



Elenco delle tabelle

Tabella 2	Requisiti derivanti dal front-end UI.	37
Tabella 3	Requisiti derivanti dal back-end UI.	55
Tabella 4	Requisiti derivanti dalla API rilevazioni.	58



Elenco delle figure

Figura 1	Diagramma dei package del sistema.	30
Figura 2	Diagramma ER della base di dati.	59
Figura 3	Diagramma delle classi della libreria di persistenza.	61
Figura 4	Diagramma di sequenza della richiesta dell'elenco delle macchine.	66
Figura 5	Diagramma di sequenza della ricezione di una rilevazione.	67
Figura 6	Diagramma delle classi dell'interceptor.	69
Figura 7	Diagramma delle classi di una guard.	70
Figura 8	Diagramma delle classi della schermata di autenticazione.	72
Figura 9	Diagramma delle classi della barra di navigazione.	74
Figura 10	Diagramma delle classi della schermata di modifica password.	76
Figura 11	Diagramma delle classi dell'albero di selezione caratteristiche.	78
Figura 12	Diagramma delle classi del componente di visualizzazione grafici.	81
Figura 13	Diagramma delle classi della schermata di gestione macchine.	84
Figura 14	Diagramma delle classi della schermata di creazione macchina.	88
Figura 15	Diagramma delle classi della schermata di modifica macchina.	91
Figura 16	Diagramma delle classi della finestra di dialogo di creazione caratteristica.	96
Figura 17	Diagramma delle classi della finestra di dialogo di modifica caratteristica.	98
Figura 18	Diagramma delle classi della finestra di lista utenti.	100
Figura 19	Diagramma delle classi relativo alla gestione delle eccezioni.	116
Figura 20	Diagramma delle classi relativo all'autenticazione.	118
Figura 21	Diagramma delle classi relativo alla modifica della password di un utente.	123
Figura 22	Diagramma delle classi relativo all'ottenimento della lista delle macchine non archiviate.	127
Figura 23	Diagramma delle classi relativo all'ottenimento della lista delle caratteristiche non archiviate di una macchina.	129
Figura 24	Diagramma delle classi relativo all'ottenimento della lista delle rilevazioni di una caratteristica.	132
Figura 25	Diagramma delle classi relativo all'ottenimento dei limiti tecnici di una caratteristica.	136
Figura 26	Diagramma delle classi relativo all'ottenimento della lista delle macchine.	138
Figura 27	Diagramma delle classi relativo all'aggiornamento dello stato di archiviazione di una macchina.	141
Figura 28	Diagramma delle classi relativo all'aggiornamento dello stato di attivazione di una macchina.	143
Figura 29	Diagramma delle classi relativo all'ottenimento dei dettagli di una macchina.	145
Figura 30	Diagramma delle classi relativo alla modifica di una macchina.	146
Figura 31	Diagramma delle classi relativo all'inserimento di una nuova macchina.	148
Figura 32	Diagramma delle classi relativo all'ottenimento della lista delle caratteristiche.	152
Figura 33	Diagramma delle classi relativo alla modifica dello stato di archiviazione di una caratteristica.	155
Figura 34	Diagramma delle classi relativo all'inserimento di una caratteristica.	158
Figura 35	Diagramma delle classi relativo alla modifica di una caratteristica.	160
Figura 36	Diagramma delle classi relativo all'ottenimento della lista degli utenti.	163
Figura 37	Diagramma delle classi relativo alla modifica dello stato di archiviazione di un utente.	165
Figura 38	Diagramma delle classi relative all'inserimento di un utente.	167
Figura 39	Diagramma delle classi relative alla modifica dell'utente per mano di un amministratore.	170
Figura 40	Diagramma delle classi relative a AccountsConfiguration.	172
Figura 41	Diagramma delle classi relative a AdminsAccountsConfiguration.	173

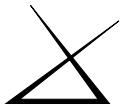


Figura 42	Diagramma delle classi relative a AdminsDevicesConfiguration	175
Figura 43	Diagramma delle classi relative a DetectionsConfiguration	178
Figura 44	Diagramma delle classi relative a DevicesConfiguration	179
Figura 45	Diagramma delle classi relative a DetectionsController	182
Figura 46	Diagramma delle classi relative a BusinessException	184
Figura 47	Diagramma delle classi relative a DetectionsAdapter	185
Figura 48	Diagramma delle classi relative a DetectionsService	192
Figura 49	Diagramma delle classi relative a DetectionValidator	194
Figura 50	Diagramma delle classi relative a DetectionQueue	196
Figura 51	Diagramma delle classi relative a DetectionSeries	198
Figura 52	Diagramma delle classi relative a SeriesPortFacade	202
Figura 53	Diagramma delle classi relative a ControlLimitsCalculator	203
Figura 54	Diagramma delle classi relative a ControlChart	205
Figura 55	Diagramma delle classi relative a DetectionsConfiguration	210
Figura 56	Diagramma di sequenza del processo di autenticazione	213
Figura 57	Diagramma di sequenza della modifica della password	214
Figura 58	Diagramma di sequenza della creazione di un grafico	217
Figura 59	Diagramma di sequenza della creazione di un grafico che mostra i punti compresi tra due estremi temporali scelti dall'utente	219
Figura 60	Diagramma di sequenza di inizializzazione componente	220
Figura 61	Diagramma di sequenza di creazione di una macchina	222
Figura 62	Diagramma di sequenza dell'aggiunta di una caratteristica a una macchina	223
Figura 63	Diagramma di sequenza della modifica di una caratteristica	226
Figura 64	Diagramma di sequenza dell'inserimento di un nuovo utente	227
Figura 65	Diagramma di sequenza della modifica di un utente	229
Figura 66	Diagramma di sequenza relativo alla modifica della password	247
Figura 67	Diagramma di sequenza relativo all'ottenimento delle macchine non archiviate	247
Figura 68	Diagramma di sequenza relativo all'ottenimento della lista delle caratteristiche non archiviate di una macchina	248
Figura 69	Diagramma di sequenza relativo all'ottenimento dei limiti di una caratteristica	248
Figura 70	Diagramma di sequenza relativo all'ottenimento dei limiti di una caratteristica	249
Figura 71	Diagramma di sequenza relativo all'ottenimento della lista delle macchine	250
Figura 72	Diagramma di sequenza relativo all'archiviazione di una macchina	250
Figura 73	Diagramma di sequenza relativo alla disattivazione di una macchina	251
Figura 74	Diagramma di sequenza relativo all'ottenimento dei dettagli di una macchina	251
Figura 75	Diagramma di sequenza relativo alla modifica di una macchina	252
Figura 76	Diagramma di sequenza relativo all'inserimento di una macchina	252
Figura 77	Diagramma di sequenza relativo all'ottenimento della lista delle caratteristiche di una macchina	253
Figura 78	Diagramma di sequenza relativo all'archiviazione di una caratteristica	254
Figura 79	Diagramma di sequenza relativo all'inserimento di una caratteristica	254
Figura 80	Diagramma di sequenza relativo alla modifica di una caratteristica	255
Figura 81	Diagramma di sequenza relativo all'ottenimento della lista degli utenti	256
Figura 82	Diagramma di sequenza relativo all'archiviazione di un utente	256
Figura 83	Diagramma di sequenza relativo all'inserimento di un utente	257
Figura 84	Diagramma di sequenza relativo alla modifica di un utente da parte di un amministratore	257
Figura 85	Diagramma di sequenza relativo alla gestione di un errore	258
Figura 86	Diagramma di sequenza relativo alla ricezione di una rilevazione	267
Figura 87	Diagramma di sequenza relativo alla gestione di un errore	268
Figura 88	Diagramma di sequenza relativo all'analisi di una rilevazione	269
Figura 89	Zone della carta di controllo	293

1 Introduzione

1.1 Scopo del documento

Lo scopo del documento è descrivere l'architettura del prodotto *Produlytics* a più livelli di dettaglio progressivi, secondo un approccio *top-down_G*. Vengono inoltre illustrate e motivate le tecnologie scelte. Una volta consolidate, le decisioni illustrate devono essere rigorosamente seguite dai Programmatori durante la fase di codifica.

1.2 Scopo del prodotto

Lo scopo è realizzare una *web application_G* per supervisionare l'andamento della produzione di alcune macchine, tramite *carte di controllo_G*. Le *macchine_G* sono configurate da un amministratore, mentre l'applicazione delle carte di controllo è visualizzabile da tutti gli utenti su una o più *caratteristiche_G* scelte.

1.3 Glossario

Onde evitare ambiguità nei termini utilizzati nel documento, viene fornito in appendice anche un *Glossario*, dove vengono definiti tutti i termini con un significato particolare.

Un termine presente nel *Glossario* viene contrassegnato dal corsivo e da una 'G' aggiunta a pedice.

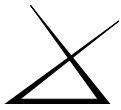
1.4 Riferimenti

1.4.1 Normativi

- *Norme di Progetto v2.0*;
- Capitolato d'appalto C3 - CC4D:
 - <https://www.math.unipd.it/~tullio/IS-1/2021/Progetto/C3p.pdf>;
 - <https://www.math.unipd.it/~tullio/IS-1/2021/Progetto/C3.pdf>.

1.4.2 Informativi

- *Analisi dei Requisiti v2.0*;
- *Piano di Qualifica v2.0*;
- Slide dell'insegnamento di Ingegneria del Software, in particolare:
 - Progettazione software:
<https://www.math.unipd.it/~tullio/IS-1/2021/Dispense/T09.pdf>;
 - UML - Diagrammi delle classi:
https://www.math.unipd.it/~rcardin/swea/2021/Diagrammi%20delle%20Classi_4x4.pdf;
 - UML - Diagrammi di sequenza:
<https://www.math.unipd.it/~rcardin/swea/2022/Diagrammi%20di%20Sequenza.pdf>;
 - Software architecture patterns:
<https://www.math.unipd.it/~rcardin/swea/2022/Software%20Architecture%20Patterns.pdf>;
 - Software creational design patterns:
<https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Creazionali.pdf>;



- Software structural design patterns:
<https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Strutturali.pdf>;
- Dependency injection:
<https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Architetturale%20-%20Dependency%20Injection.pdf>.
- Software Engineering - Ian Sommerville (10th Edition), in particolare:
 - Chapter 5: system modeling;
 - Chapter 6: architectural design.
- StarUML: <https://staruml.io/>;
- PostgreSQL: <https://www.postgresql.org/docs/14/index.html>;
- Timescale: <https://docs.timescale.com/>;
- Java: <https://docs.oracle.com/en/java/javase/17/docs/api/index.html>;
- Spring Boot: <https://spring.io/projects/spring-boot>;
- Spring Security: <https://spring.io/guides/topicals/spring-security-architecture>;
- Spring Data JPA: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference>;
- Spring Data REST: <https://docs.spring.io/spring-data/rest/docs/current/reference/html/#reference>;
- TypeScript: <https://www.typescriptlang.org/docs/>;
- Angular: <https://angular.io/docs>;
- D3.js: <https://github.com/d3/d3/wiki>;
- REST: <https://restfulapi.net>;
- JSON: <https://www.json.org/json-en.html>;
- Docker: <https://docs.docker.com/>;
- Docker Compose: <https://docs.docker.com/compose/>.

2 Tecnologie utilizzate

Lo scopo di questa sezione è elencare e descrivere le tecnologie utilizzate nello sviluppo del *progetto_G* *Produlytics*, illustrando anche i vantaggi e gli svantaggi che sono stati valutati in fase di scelta e che le hanno favorite a discapito di altre tecnologie proposte in alternativa dal proponente.

2.1 PostgreSQL

PostgreSQL_G è un *DBMS_G* relazionale e *open source_G*, che utilizza il linguaggio *SQL_G* per le *query_G*. Le alternative scartate sono: *MySQL_G*, *MariaDB_G* e *MongoDB*.

Vantaggi:

- essendo un *database_G* basato su SQL dispone di tutte le caratteristiche tipiche dei database relazionali come il pieno supporto ai vincoli di integrità e alle *transazioni_G*;
- tutto il gruppo ha esperienza con PostgreSQL, riducendo notevolmente i tempi di formazione;
- è uno dei database relazionali più utilizzati, ne consegue facilità nel trovare documentazione aggiuntiva e supporto.

Svantaggi:

- tende ad avere performance peggiori rispetto ad alternative *NoSQL_G* come MongoDB.

Versione scelta: 14.

2.2 Timescale

Timescale_G è un *time series_G* database open source basato su SQL, creato come estensione alle funzionalità di PostgreSQL.

Le alternative scartate sono: *MariaDB ColumnStore_G*, *ClickHouse_G* e *MongoDB_G*.

Vantaggi:

- essendo un'estensione di PostgreSQL i tempi di formazione sono modesti;
- è basato su SQL, con tutti i vantaggi precedentemente descritti;
- è ottimizzato in modo specifico per operare con dati time series, che verranno ampiamente utilizzati nel progetto;
- è uno dei database time series più utilizzati, pertanto è presente una community di supporto molto espansa.

Svantaggi:

- come per PostgreSQL, tende ad avere performance peggiori rispetto ad alternative NoSQL come MongoDB.

Versione scelta: 2.6.0.

2.3 Java

$Java_G$ è un linguaggio *general purpose* $_G$, orientato agli oggetti e staticamente tipizzato. È pensato per essere indipendente dalla piattaforma di esecuzione sottostante.

L'alternativa scartata è $JavaScript_G$.

Vantaggi:

- è uno dei linguaggi più usati al mondo, ne consegue facilità nel trovare documentazione aggiuntiva e supporto;
- alcuni componenti del gruppo ne hanno già familiarità;
- fornisce supporto per il *multithreading* $_G$.

Versione scelta: 17.

2.4 Spring Boot

$Spring\ Boot_G$ è un *framework* $_G$ Java ampiamente utilizzato per lo sviluppo di web application ma anche per l'implementazione di API_G $REST_G$ in tempi rapidi.

L'alternativa scartata è $Express_G$.

Vantaggi:

- è basato su $Spring_G$, il framework Java più utilizzato al mondo, ne consegue facilità nel trovare documentazione aggiuntiva e supporto;
- porta a una riduzione del codice *boilerplate* $_G$;
- permette una facile comunicazione con il database.

Versione scelta: 2.6.2.

2.5 TypeScript

$TypeScript_G$ è un superset di JavaScript, che lo estende aggiungendo funzionalità opzionali come la tipizzazione statica e il supporto alle interfacce.

Vantaggi:

- la tipizzazione statica aiuta a prevenire errori a runtime;
- è un linguaggio popolare, ne consegue facilità nel trovare documentazione aggiuntiva e supporto.

Versione scelta: 4.

2.6 Angular

$Angular_G$ è un framework open source per lo sviluppo di web application, caratterizzato dall'utilizzo del linguaggio TypeScript.

Le alternative scartate sono: $React_G$ e Vue_G .

Vantaggi:

- un'applicazione Angular è basata su componenti che semplificano poi il testing;
- è adatto alla creazione di PWA_G .

Svantaggi:

- la curva di apprendimento è piuttosto ripida.

Versione scelta: 13.

2.7 D3.js

$D3.js_G$ è una libreria JavaScript per creare visualizzazioni di dati dinamiche e interattive su $browser_G$. L'adozione di questa tecnologia è vincolata dalle richieste del proponente.

Versione scelta: 7.

2.8 REST

Per la progettazione della API si è scelto un approccio REST. L'alternativa scartata è $GraphQL_G$.

Vantaggi:

- REST è uno stile architetturale, al contrario di GraphQL che è un linguaggio. Ciò implica tempi di formazione minori;
- è ampiamente supportato dai framework più recenti;
- è facilmente scalabile.

Svantaggi

- non previene *overfetching_G* e *underfetching_G*, al contrario di GraphQL. Notare, però, che l'eventualità di questi fenomeni è minima, dato il numero limitato di entità del sistema.

2.9 JSON

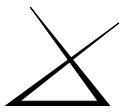
$JSON_G$ è il formato dati scelto per lo scambio di dati fra $client_G$ e $server_G$. L'unica tecnologia alternativa presa in considerazione in questo ambito è stata XML_G , scartata in quanto JSON si integra più facilmente con JavaScript e i framework moderni.

2.10 Docker

$Docker_G$ è un framework per il rilascio di applicazioni all'interno di container. Un container, similmente a una macchina virtuale, nasconde le caratteristiche della macchina sottostante e offre vantaggi di portabilità e sicurezza. L'adozione di questa tecnologia è vincolata dalle richieste del proponente.

Vantaggi:

- l'esecuzione di un container è isolata e di conseguenza molto sicura, oltre a questo è possibile aumentare il livello di protezione attraverso diverse impostazioni di esecuzione;
- un container, rispetto a una macchina virtuale, ha un contesto di esecuzione più leggero, dato dall'assenza di un sistema operativo proprio;
- un'immagine docker può essere eseguita in container di una qualsiasi macchina;
- il framework supporta un versionamento delle immagini e un'efficace funzione di ripristino a versioni precedenti.

**Svantaggi:**

- l'isolamento dei processi di un container non è mai perfetto, poiché questi vengono eseguiti sul sistema operativo della macchina sottostante.

Versione scelta: 20.10.11.

2.11 Docker Compose

*Docker Compose*_G è uno strumento per facilitare la gestione e l'esecuzione di più container docker, operando su semplici file *YAML*_G di configurazione.

Vantaggi:

- la sintassi di definizione è semplice;
- permette di gestire più container tramite una singola interfaccia;
- ogni sua funzione può essere replicata dalla normale interfaccia di Docker.

Versione scelta: 2.2.3.

3 Architettura di sistema

La descrizione del sistema adotta un approccio top-down, partendo dalla struttura generale per poi scendere nel dettaglio.

Il sistema deve offrire due servizi diversi, che condividono una stessa banca dati:

1. un'interfaccia web;
2. un'API rilevazioni.

3.1 Modello architetturale

Il sistema è progettato seguendo come modello un'architettura *client-server_G* three-tier ed è separato in tre strati, logicamente e fisicamente divisi, che si identificano nelle seguenti componenti:

- **strato di presentazione:**
 - un front-end UI, per l'interfaccia web;
 - per l'API rilevazioni non viene fornito un client.
- **strato di applicazione:**
 - un back-end UI, per l'interfaccia web;
 - una componente API rilevazioni, per l'omonimo servizio.
- **strato di persistenza:** la banca dati, condivisa da entrambi i servizi.

Seguendo il modello client-server, lo strato di presentazione non può comunicare direttamente con lo strato di persistenza, dovendo passare sempre per lo strato di applicazione.

Questo modello di architettura è stato scelto perché permette una separazione netta tra i vari strati e quindi la loro progettazione e sviluppo in parallelo.

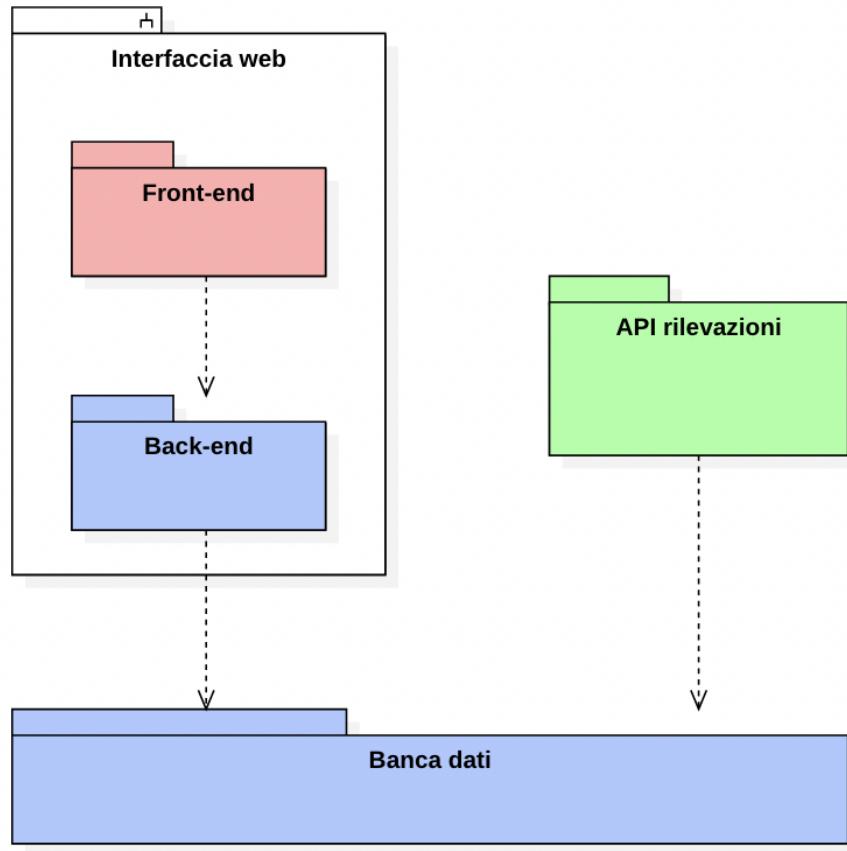


Figura 1: Diagramma dei package del sistema.

3.2 Descrizione delle componenti

Partendo dall'architettura generale del sistema vengono identificate le seguenti quattro componenti:

1. il front-end UI, che si occupa di:
 - costruire la parte grafica dell'applicazione, a partire dai dati richiesti al back-end UI;
 - permettere all'utente l'interazione con l'applicazione.
2. il back-end UI, che si occupa di:
 - elaborare le richieste degli utenti, verificando la presenza di una corretta autenticazione da parte degli stessi e i loro permessi;
 - effettuare richieste di lettura o scrittura verso la banca dati.
3. la API rilevazioni, che fornisce un'interfaccia da cui il sistema può ricevere le rilevazioni delle macchine. Segue uno stile REST e si occupa di portare le rilevazioni dalle macchine alla banca dati, seguendo i seguenti passi:
 - (a) riceve i dati provenienti dalle macchine configurate nell'applicazione;
 - (b) i dati vengono elaborati dal motore di calcolo, per capire se sono presenti dei valori fuori scala utilizzando le carte di controllo;

- (c) i dati elaborati vengono memorizzati nella tabella delle rilevazioni della banca dati.
4. la banca dati, che è divisa in due parti:
- le tabelle di configurazione, che memorizzano tutti i dati relativi agli utenti del sistema e alla loro gestione, le informazioni sulle macchine censite e sulle loro caratteristiche;
 - la tabella delle rilevazioni, che memorizza le rilevazioni ricevute dalle macchine.

3.3 Assemblaggio delle componenti

Le componenti sono assemblate insieme utilizzando Docker Compose. In particolare sono prodotti i seguenti container Docker:

- **db**: contiene la componente di banca dati. Non espone porte all'esterno, ma la banca dati è accessibile dagli altri container attraverso l'indirizzo db:5432;
- **web-ui**: contiene le componenti back-end UI e front-end UI. Espone la porta 80 all'esterno, attraverso cui avvengono le richieste sia verso il back-end UI che il front-end UI. Il back-end UI si deve quindi occupare anche di servire le risorse statiche del front-end UI sotto il prefisso /#/;
- **api**: contiene la componente API rilevazioni. Espone la porta 81 all'esterno per permettere alle macchine di inviare le rilevazioni.

3.4 Configurazione di sistema

3.4.1 Front-end UI

3.4.1.1 Definizione dell'interfaccia

3.4.1.1.1 Schermata di autenticazione

Questa schermata permette all'utente di autenticarsi ed è automaticamente saltata se ha una sessione memorizzata. In particolare, è costituita da:

- **percorso**: /login;
- **elementi**:
 - una casella di testo per il nome utente;
 - una casella di testo per la password;
 - una checkbox per richiedere la memorizzazione della sessione;
 - un pulsante per effettuare un tentativo di login;
 - in seguito a un tentativo di login fallito, un messaggio di errore.

Essa soddisfa i requisiti funzionali: ROF1, ROF1.1, ROF1.1.1, ROF1.1.2, RDF1.1.3, ROF1.1.4, ROF1.1.5, ROF1.1.5.1.

3.4.1.1.2 Barra di navigazione

La barra di navigazione è l'elemento di riferimento per muoversi all'interno dell'applicazione, viene condivisa da tutte le schermate a eccezione della schermata di autenticazione (§3.4.1.1.1). In particolare, è costituita da:

- **elementi**:
 - un pulsante per accedere alla schermata di monitoraggio rilevazioni (§3.4.1.1.5);

- un menù contestuale a comparsa composto da:
 - * se l'utente autenticato è un amministratore, un collegamento per accedere alla schermata di gestione macchine (§3.4.1.1.6);
 - * se l'utente autenticato è un amministratore, un collegamento per accedere alla schermata di gestione utenti (§3.4.1.1.11);
 - * un collegamento per accedere alla schermata di modifica della propria password (§3.4.1.1.4);
 - * un collegamento per eseguire il logout e passare alla schermata di autenticazione (§3.4.1.1.1).

Essa soddisfa i requisiti funzionali: ROF2, ROF3, ROF13, ROF14, ROF15, ROF16, ROF17, ROF18, RO19.

3.4.1.1.3 Finestra di dialogo di conferma

La finestra di dialogo di conferma si occupa di gestire la conferma di un'operazione potenzialmente pericolosa. In particolare, è costituita da:

- **percorso:** finestra di dialogo all'interno di /{schermata}, dove:
 - {schermata} è il nome della schermata da cui è stata generata l'operazione potenzialmente pericolosa.
- **elementi:**
 - la descrizione dell'operazione in corso;
 - un pulsante per annullare l'operazione in corso;
 - un pulsante per confermare l'operazione in corso.

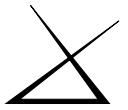
Essa soddisfa i requisiti funzionali: ROF14.4.1.3, ROF16.1.3, ROF19.1.2.

3.4.1.1.4 Schermata di modifica password

Questa schermata è una finestra di dialogo che permette all'utente di modificare la propria password o che glielo impedisce se commette degli errori. È accessibile dalla barra di navigazione (§3.4.1.1.2). In particolare, è costituita da:

- **percorso:** finestra di dialogo all'interno di /;
- **elementi:**
 - una casella di testo dove inserire la password corrente;
 - una casella di testo dove inserire la nuova password;
 - una casella di testo dove reinserire la nuova password;
 - un pulsante per confermare la modifica della password;
 - in seguito a un errato inserimento della password attuale, un messaggio di errore;
 - in seguito a un inserimento di una nuova password non valida, un messaggio di errore;
 - in seguito a un inserimento non combaciante della ripetizione della nuova password, un messaggio di errore.

Essa soddisfa i requisiti funzionali: ROF3, ROF3.1, ROF3.2, ROF3.3, ROF3.4, ROF3.5, ROF3.5.1, RPF3.6, RPF3.6.1, RPF3.6.2, ROF3.7, ROF3.7.1.



3.4.1.1.5 Schermata di monitoraggio rilevazioni

Questa schermata permette all'utente di scegliere le macchine e le caratteristiche da monitorare e di visualizzarne i grafici. È accessibile dalla barra di navigazione (§3.4.1.1.2). In particolare, è costituita da:

- **percorso:** /;
- **elementi:**
 - un menù ad albero a due livelli, dal quale è possibile scegliere le macchine non archiviate e le loro caratteristiche non archiviate;
 - un pulsante per confermare le scelte effettuate che, una volta premuto, rende visibile un pannello contenente:
 - * i grafici relativi alle scelte effettuate, aggiornati in modo automatico ogni secondo;
 - * un pulsante per visualizzare i grafici in modalità carosello;
 - * in modalità carosello, dei tasti che permettano di passare al grafico precedente o a quello successivo;
 - * in ogni grafico:
 - una barra in cui è possibile selezionare la porzione di grafico da visualizzare;
 - una retta che rappresenta la media;
 - una retta per ogni limite;
 - un punto nero per ogni rilevazione;
 - un punto rosso per ogni rilevazione anomala.

I grafici nel pannello di monitoraggio aggiornano la propria rappresentazione ogni 5 secondi. Essa soddisfa i requisiti funzionali: ROF4, ROF4.1, ROF4.1.1, ROF4.1.2, ROF4.1.3, ROF4.2, ROF4.3, ROF5, ROF5.1, ROF6, RDF7, RDF7.1, ROF8, ROF9, ROF10, ROF14.4.1.2, ROF14.4.2.2, ROF16.1.2, ROF16.2.2.

3.4.1.1.6 Schermata di gestione macchine

Questa schermata permette all'amministratore di:

- accedere alla schermata di inserimento macchina;
- accedere alla schermata di modifica macchina;
- modificare lo stato di una macchina;
- archiviare o ripristinare una macchina.

In particolare, è costituita da:

- **percorso:** /gestione-macchine;
- **elementi:**
 - una lista di tutte le macchine, in cui ogni riga contiene:
 - * il nome della macchina;
 - * un pulsante per modificare la macchina (§3.4.1.1.8);
 - * se la macchina non è disattivata, un pulsante per disattivarla;
 - * se la macchina è disattivata, un pulsante per attivarla;
 - * se la macchina non è archiviata, un pulsante per archiviarla che una volta cliccato apre una finestra di dialogo di conferma (§3.4.1.1.3);

- * se la macchina è archiviata, un pulsante per ripristinare la macchina.
- un pulsante per l'inserimento di una macchina (§3.4.1.1.7).

Essa soddisfa i requisiti funzionali: ROF15, ROF15.1, ROF15.2, ROF16, ROF16.1, ROF16.1.3, ROF16.2, ROF21.

3.4.1.1.7 Schermata di creazione macchina

Questa schermata permette all'amministratore di creare una nuova macchina o che glielo impedisce se i dati inseriti non sono validi. È accessibile dalla schermata di gestione delle macchine (§3.4.1.1.6). In particolare, è costituita da:

- **percorso:** /gestione-macchine/nuova;
- **elementi:**
 - una casella di testo dove inserire il nome della nuova macchina;
 - un pulsante per inserire una nuova caratteristica nella macchina;
 - un pulsante per confermare la creazione della macchina;
 - in seguito a un inserimento di dati non validi, un messaggio di errore;
 - in seguito a un mancato inserimento di almeno una caratteristica, un messaggio di errore.

Essa soddisfa i requisiti funzionali: ROF13, ROF13.1, ROF13.2, ROF13.3, ROF13.4.1, ROF13.5, ROF13.5.1, ROF13.6, ROF13.6.1.

3.4.1.1.8 Schermata di modifica macchina

Questa schermata visualizza i dettagli di una macchina e ne permette la modifica. È raggiungibile dalla schermata di gestione delle macchine (§3.4.1.1.6). In particolare, è costituita da:

- **percorso:** /gestione-macchine/{idMacchina}, dove:
 - {idMacchina} è il codice identificativo della macchina.
- **elementi:**
 - le informazioni della macchina:
 - * il nome;
 - * il token di accesso per l'API rilevazioni;
 - * se è archiviata o meno;
 - * se è disattivata o meno.
 - un pulsante e una casella di testo che permettono di modificare il nome della macchina;
 - in seguito all'inserimento del nome di una macchina già esistente, un messaggio di errore;
 - un elenco delle caratteristiche della macchina. Dalle voci di questo elenco sono disponibili:
 - * un pulsante per raggiungere la schermata per effettuare modifiche alla caratteristica (§3.4.1.1.10);
 - * un pulsante che permette di archiviare o ripristinare la caratteristica, previa richiesta esplicita di conferma, in base allo stato corrente.
 - un pulsante che permette l'aggiunta di una nuova caratteristica (§3.4.1.1.10).

Essa soddisfa i requisiti funzionali: ROF14, ROF14.1, ROF14.2, ROF14.2.1, ROF14.2.1.1, ROF14.2.2, ROF14.2.2.1, ROF14.3, ROF14.4, ROF14.4.1, ROF14.4.1.3, ROF14.4.2, ROF14.5, ROF14.6, ROF14.6.1, ROF14.7, ROF14.7.1.

3.4.1.1.9 Form per la creazione di una caratteristica

Questo form permette di inserire una nuova caratteristica in una macchina ed è accessibile dalle schermate di creazione (§3.4.1.1.7) e di modifica della macchina (§3.4.1.1.8). In particolare, è costituito da:

- **percorso:** /gestione-macchine/{idMacchina}, dove:
 - {idMacchina} è il codice identificativo della macchina.
- **elementi:**
 - una casella di testo per inserire il nome della caratteristica;
 - una casella di testo per inserire il *limite tecnico_G* inferiore della caratteristica;
 - una casella di testo per inserire il limite tecnico superiore della caratteristica;
 - una checkbox per impostare l'auto-adjust, se attivato abilita una casella di testo per scrivere la grandezza del campione dei dati utilizzati dal motore di calcolo;
 - un pulsante per confermare l'inserimento della caratteristica e chiudere il form;
 - un pulsante per annullare l'inserimento e chiudere il form;
 - in seguito all'inserimento di un nome già presente, un messaggio di errore;
 - in seguito all'inserimento di dati non validi, un messaggio di errore.

Esso soddisfa i requisiti funzionali: ROF13.2, ROF13.2.1, ROF13.2.2, ROF13.2.3, ROF13.2.4, ROF13.2.4.1, ROF13.2.5, ROF13.2.6, ROF13.2.6.1, ROF13.2.7, ROF13.2.7.1, ROF13.2.8.

3.4.1.1.10 Form per la modifica di una caratteristica

Questo form permette di modificare una caratteristica in una macchina ed è accessibile dalle schermate di inserimento (§3.4.1.1.7) e di modifica della macchina (§3.4.1.1.8). In particolare, è costituito da:

- **percorso:** /gestione-macchine/{idMacchina}, dove:
 - {idMacchina} è il codice identificativo della macchina.
- **elementi:**
 - una casella di testo per modificare il nome della caratteristica;
 - una casella di testo per modificare il limite tecnico inferiore della caratteristica;
 - una casella di testo per modificare il limite tecnico superiore della caratteristica;
 - una checkbox per impostare l'auto-adjust, se attivato abilita la casella di testo per modificare la grandezza del campione precedente;
 - un pulsante per confermare le modifiche effettuate e chiudere il form;
 - un pulsante per annullare la modifica e chiudere il form;
 - in seguito all'inserimento di un nome già presente, un messaggio di errore;
 - in seguito all'inserimento di dati non validi, un messaggio di errore.

Esso soddisfa i requisiti funzionali: ROF14.3, ROF14.3.1, ROF14.3.2, ROF14.3.3, ROF14.3.4, ROF14.3.4.1, ROF14.3.5, ROF14.3.6, ROF14.3.6.1, ROF14.3.7, ROF14.3.7.1, ROF14.3.8, ROF14.3.8.1.

3.4.1.1.11 Schermata di gestione utenti

Questa schermata permette all'amministratore di vedere tutti gli utenti e per ognuno di essi accedere alla schermata di modifica utente e archiviarlo o ripristinarlo. È accessibile dalla barra di navigazione dell'amministratore (§3.4.1.1.2). In particolare, è costituita da:

- **percorso:** /gestione-utenti;
- **elementi:**
 - una lista di tutti gli utenti, in cui ogni riga contiene:
 - * nome utente;
 - * se è amministratore, un indicatore apposito;
 - * un pulsante per modificare l'utente, che rimanda alla schermata descritta nella sezione (§3.4.1.1.13);
 - * se non è archiviato, un pulsante per archiviare, che una volta cliccato apre una finestra di dialogo di conferma (§3.4.1.1.3);
 - * se è archiviato, un pulsante per ripristinarlo.

Essa soddisfa i requisiti funzionali: ROF17, ROF19, ROF19.1, ROF19.1.2, ROF19.2.

3.4.1.1.12 Form di creazione utente

Questo form è una finestra di dialogo all'interno della schermata di gestione degli utenti (§3.4.1.1.11), che permette l'inserimento dei dati necessari alla creazione di un utente. È accessibile dalla schermata di gestione utenti (§3.4.1.1.11). In particolare, è costituito da:

- **percorso:** /gestione-utenti;
- **elementi:**
 - una casella di testo per il nome dell'utente;
 - una casella di testo per la password dell'utente;
 - un checkbox per impostare i privilegi di amministratore per l'utente;
 - un pulsante per confermare e salvare i dati inseriti;
 - in seguito all'inserimento di un nome utente già esistente, un messaggio di errore;
 - in seguito all'inserimento di una password non valida, un messaggio di errore.

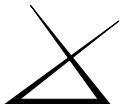
Esso soddisfa i requisiti funzionali: ROF17, ROF17.1, ROF17.2, ROF17.3, ROF17.4, ROF17.5, ROF17.5.1, RPF17.6, RPF17.6.1.

3.4.1.1.13 Form di modifica utente

Questo form è una finestra di dialogo accessibile dalla schermata di gestione degli utenti (§3.4.1.1.11), che permette la modifica della password e dei privilegi di un utente. In particolare, è costituito da:

- **percorso:** /gestione-utenti;
- **elementi:**
 - una casella di testo per la nuova password dell'utente;
 - una checkbox per impostare i privilegi di amministratore per l'utente;
 - un pulsante per confermare e salvare i dati inseriti;
 - in seguito all'inserimento di una password non valida, un messaggio di errore.

Esso soddisfa i requisiti funzionali: ROF18, ROF18.1, ROF18.2, ROF18.3, ROF18.4, ROF18.4.1.



3.4.1.2 Requisiti per i livelli inferiori

Tabella 2: Requisiti derivanti dal front-end UI.

Codice	Descrizione	Fonte
RIF1	<p>Il front-end deve poter creare una nuova sessione, dati:</p> <ul style="list-style-type: none">• il nome utente;• la password;• se la sessione deve essere ricordata o meno. <p>e ricevere il token della sessione o un codice di errore. Tipo di sessione richiesta: nessuna.</p>	§3.4.1.1.1
RIF2	<p>Il front-end deve poter effettuare il logout dalla sessione attuale. Tipo di sessione richiesta: utente.</p>	§3.4.1.1.2
RIF3	<p>Il front-end, dato l'username dell'utente, deve poterne modificare la password, trasmettendo:</p> <ul style="list-style-type: none">• la password attuale;• la nuova password. <p>e ricevere un codice di errore indicante se:</p> <ul style="list-style-type: none">• la password attuale non è corretta;• la nuova password non è valida.	§3.4.1.1.4
RIF4	<p>Il front-end deve poter ottenere la lista di tutte le macchine non archiviate e, per ognuna di esse:</p> <ul style="list-style-type: none">• l'identificativo;• il nome. <p>Tipo di sessione richiesta: utente o amministratore.</p>	§3.4.1.1.5
RIF5	<p>Il front-end deve poter ottenere, dato l'identificativo univoco di una macchina, la lista di tutte le caratteristiche non archiviate relative a quella macchina e, per ognuna di esse:</p> <ul style="list-style-type: none">• l'identificativo univoco nel contesto della macchina;• il nome. <p>Tipo di sessione richiesta: utente o amministratore.</p>	§3.4.1.1.5

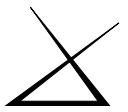


Tabella 2 (continuazione): Requisiti derivanti dal front-end UI.

Codice	Descrizione	Fonte
RIF6	<p>Il front-end deve poter ottenere, dato l'identificativo di una macchina e di una relativa caratteristica, tutte le recenti rilevazioni di quella caratteristica e quelle che verranno rilevate di secondo in secondo e, per ognuna di esse:</p> <ul style="list-style-type: none">• le coordinate temporali in cui la rilevazione è stata creata;• il valore della rilevazione;• se la rilevazione è anomala o meno. <p>Tipo di sessione richiesta: utente o amministratore.</p>	§3.4.1.1.5
RIF7	<p>Il front-end deve poter ottenere, dato l'identificativo di una macchina e di una relativa caratteristica:</p> <ul style="list-style-type: none">• il <i>limite di riferimento</i>_G inferiore;• il limite di riferimento superiore;• la media dei due limiti. <p>Tipo di sessione richiesta: utente o amministratore.</p>	§3.4.1.1.5
RIF8	<p>Il front-end deve poter ottenere la lista delle macchine e, per ognuna di esse:</p> <ul style="list-style-type: none">• il nome;• il codice identificativo;• se è disattivata o meno;• se è archiviata o meno. <p>Tipo di sessione richiesta: amministratore.</p>	§3.4.1.1.6
RIF9	Il front-end deve poter archiviare una macchina disponibile, dato l'identificativo.	§3.4.1.1.6
RIF10	Il front-end deve poter ripristinare una macchina archiviata, dato l'identificativo.	§3.4.1.1.6
RIF11	Il front-end deve poter attivare una macchina disattivata, dato l'identificativo.	§3.4.1.1.6
RIF12	Il front-end deve poter disattivare una macchina attivata, dato l'identificativo.	§3.4.1.1.6

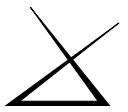


Tabella 2 (continuazione): Requisiti derivanti dal front-end UI.

Codice	Descrizione	Fonte
RIF13	<p>Il front-end deve poter aggiungere informazioni su una nuova macchina. Nello specifico:</p> <ul style="list-style-type: none">• il nome;• le sue caratteristiche, ognuna comprendente:<ul style="list-style-type: none">– il nome;– i limiti tecnici;– se è impostato l'auto-adjust;– la grandezza del campione di dati. <p>e ricevere un codice di errore indicante se:</p> <ul style="list-style-type: none">• il nome della macchina è già presente;• non è stata aggiunta nessuna caratteristica;• i dati di una caratteristica non sono validi.	§3.4.1.1.7
RIF14	<p>Il front-end deve poter ottenere, dato l'identificativo di una macchina:</p> <ul style="list-style-type: none">• il nome;• il token di accesso;• se è archiviata o meno;• se è disattivata o meno. <p>Tipo di sessione richiesta: amministratore.</p>	§3.4.1.1.8
RIF15	<p>Il front-end deve poter effettuare un aggiornamento del nome di una macchina dato il suo identificativo e ricevere un codice di errore nel caso in cui il nome esista già.</p> <p>Tipo di sessione richiesta: amministratore.</p>	§3.4.1.1.8
RIF16	<p>Il front-end deve poter elencare le caratteristiche di una macchina dato il suo identificativo e per ognuna di esse:</p> <ul style="list-style-type: none">• l'identificativo;• il nome. <p>Tipo di sessione richiesta: amministratore.</p>	§3.4.1.1.8
RIF17	<p>Il front-end deve poter archiviare una caratteristica disponibile, dati l'identificativo della macchina e l'identificativo della caratteristica.</p>	§3.4.1.1.8
RIF18	<p>Il front-end deve poter ripristinare una caratteristica archiviata, dati l'identificativo della macchina e l'identificativo della caratteristica.</p>	§3.4.1.1.8

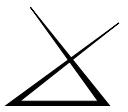


Tabella 2 (continuazione): Requisiti derivanti dal front-end UI.

Codice	Descrizione	Fonte
RIF19	<p>Il front-end deve poter aggiungere le informazioni di una nuova caratteristica, dati:</p> <ul style="list-style-type: none">• il nome;• i limiti tecnici inferiore e superiore;• se è impostato l'auto-adjust;• la grandezza del campione di dati. <p>e ricevere un codice di errore indicante se:</p> <ul style="list-style-type: none">• il nome esiste già;• i dati inseriti non sono validi. <p>Tipo di sessione richiesta: amministratore.</p>	§3.4.1.1.9
RIF20	<p>Il front-end deve poter modificare le informazioni di una caratteristica. Nello specifico:</p> <ul style="list-style-type: none">• il nome della caratteristica;• i limiti tecnici inferiore e superiore della caratteristica;• se è impostato l'auto-adjust;• la grandezza del campione di dati. <p>e ricevere un codice di errore indicante se:</p> <ul style="list-style-type: none">• il nome esiste già;• i dati inseriti non sono validi. <p>Tipo di sessione richiesta: amministratore.</p>	§3.4.1.1.10
RIF21	<p>Il front-end deve poter ottenere la lista degli utenti e, per ognuno di essi:</p> <ul style="list-style-type: none">• il nome utente;• se è amministratore o meno;• se è archiviato o meno. <p>Tipo di sessione richiesta: amministratore.</p>	§3.4.1.1.11
RIF22	Il front-end deve poter archiviare un utente disponibile, dato il nome utente.	§3.4.1.1.11
RIF23	Il front-end deve poter ripristinare un utente archiviato, dato il nome utente.	§3.4.1.1.11

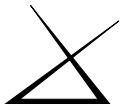
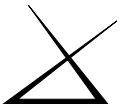


Tabella 2 (continuazione): Requisiti derivanti dal front-end UI.

Codice	Descrizione	Fonte
RIF24	<p>Il front-end deve poter aggiungere le informazioni di un nuovo utente, dati:</p> <ul style="list-style-type: none">• il nome utente;• la password dell'utente;• se l'utente possiede i privilegi di amministratore o meno. <p>e ricevere un codice di errore se:</p> <ul style="list-style-type: none">• l'utente esiste già;• la password non rispetta i requisiti. <p>Tipo di sessione richiesta: amministratore.</p>	§3.4.1.1.12
RIF25	<p>Il front-end deve poter modificare le informazioni di un utente dato il suo nome utente. Nello specifico:</p> <ul style="list-style-type: none">• la password dell'utente;• se l'utente possiede i privilegi di amministratore o meno. <p>e ricevere un codice di errore se la password non rispetta i requisiti.</p> <p>Tipo di sessione richiesta: amministratore.</p>	§3.4.1.1.13



3.4.2 Back-end UI

3.4.2.1 Definizione dell'interfaccia

3.4.2.1.1 Autenticazione

Per permettere l'identificazione dell'utente, a ogni richiesta $HTTP_G$, il client invia al server il cookie di sessione "PRODULYTICS_S". Per implementare la funzionalità "ricordami" viene utilizzato un cookie distinto da quello di sessione, chiamato "PRODULYTICS_RM", anch'esso viene inviato al server a ogni richiesta HTTP.

Nel caso in cui il cookie non sia valido, viene restituito:

- **codice di stato $HTTP_G$:** 401;
- **body:** { "errorCode": "unauthorized" }.

Nel caso in cui invece un utente non abbia i permessi per visualizzare una pagina, ma vi arrivi tramite l'URL, esso varrà reindirizzato alla pagina di login.

3.4.2.1.2 Login

Questo endpoint REST permette al front-end UI di verificare la validità del tentativo di autenticazione da parte di un utente. Consiste in:

- **endpoint:** /login;
- **metodo HTTP:** POST;
- **body:** formato JSON, un oggetto con i seguenti campi:
 - "username": una stringa rappresentante l'username dell'utente;
 - "password": una stringa rappresentante la password dell'utente;
 - "rememberMe": un booleano indicante se la sessione deve essere memorizzata.

In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 200;
- **header:**
 - se `rememberMe="true"`: Set-Cookie: PRODULYTICS_RM=<cookie-value>; Expires=<date>, dove:
 - * <cookie-value> è una stringa composta da un hash dell'username, la password, la scadenza e una chiave segreta del back-end;
 - * <date> è una data rappresentante la data di scadenza del cookie, pari a due settimane dalla data di creazione dello stesso.
- **body:** vuoto.

Se il nome utente non esiste o la password non è corretta, viene restituito:

- **codice di stato $HTTP$:** 401;
- **body:** { "errorCode": "unauthorized" }.

Esso soddisfa il requisito del front-end UI: RIF1.

3.4.2.1.3 Logout

Questo endpoint REST, implementato di default da Spring Security, permette al front-end UI di terminare la sessione di un utente. Consiste in:

- **endpoint:** /logout;
- **metodo HTTP:** POST;
- **body:** vuoto.

In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 200;
- **body:** vuoto.

Esso soddisfa il requisito del front-end UI: RIF2.

3.4.2.1.4 Modifica password

Questo endpoint REST permette al front-end UI di modificare la password dell'utente corrente. Consiste in:

- **endpoint:** /accounts/{username}/password, dove:
 - {username} è il nome dell'utente che sta effettuando l'operazione.
- **metodo HTTP:** PUT;
- **body:** formato JSON, un oggetto con i seguenti campi:
 - "currentPassword": una stringa rappresentante la password attuale che l'utente vuole modificare;
 - "newPassword": una stringa rappresentante la nuova password che l'utente vuole impostare.

In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 204;
- **body:** vuoto.

In caso di esito negativo, viene restituito:

- se la password attuale non è corretta:
 - * **codice di stato HTTP:** 401;
 - * **body:** { "errorCode": "wrongCurrentPassword" }.
- se la password nuova non è valida:
 - * **codice di stato HTTP:** 400;
 - * **body:** { "errorCode": "invalidNewPassword" }.

Esso soddisfa il requisito del front-end UI: RIF3.

3.4.2.1.5 Lista macchine non archiviate

Questo endpoint REST permette al front-end UI di ottenere la lista di macchine non archiviate per un utente. Consiste in:

- **endpoint:** /devices;
- **metodo HTTP:** GET;
- **body:** vuoto.

Viene restituito:

- **codice di stato HTTP:** 200;
- **body:** formato JSON, una lista di oggetti con i seguenti campi:
 - "id": un numero intero rappresentante l'identificativo della macchina;
 - "name": una stringa rappresentante il nome della macchina.

Esso soddisfa il requisito del front-end UI: RIF4.

3.4.2.1.6 Lista caratteristiche non archiviate di una macchina

Questo endpoint REST permette al front-end UI di ottenere la lista di caratteristiche non archiviate di una macchina per un utente. Consiste in:

- **endpoint:** /devices/{deviceId}/characteristics, dove:
 - {deviceId} è l'identificativo della macchina.
- **metodo HTTP:** GET;
- **body:** vuoto.

In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 200;
- **body:** formato JSON, una lista di oggetti con i seguenti campi:
 - "id": un numero intero rappresentante l'identificativo della caratteristica;
 - "name": una stringa rappresentante il nome della caratteristica.

In caso di esito negativo, viene restituito:

- **codice di stato HTTP:** 404;
- **body:** { "errorCode": "deviceNotFound" }

Esso soddisfa il requisito del front-end UI: RIF5.

3.4.2.1.7 Rilevazioni di una caratteristica

Questo endpoint REST permette al front-end UI di ottenere le rilevazioni relative a una caratteristica per un utente. Consiste in:

- **endpoint:** `/devices/{deviceId}/characteristics/{characteristicId}`
`/detections?olderThan={olderThan}&newerThan={newerThan}&limit={maxNumDetections}`,
dove:
 - `{deviceId}` è l'identificativo della macchina;
 - `{characteristicId}` è l'identificativo della caratteristica;
 - `{olderThan}` indica l'istante di tempo come millisecondi trascorsi dall'*UNIX epoch_G* rispetto al quale le rilevazioni ottenute dovranno essere meno recenti. Non può essere usato insieme a `newerThan`;
 - `{newerThan}` indica l'istante di tempo come millisecondi trascorsi dall'*UNIX epoch* rispetto al quale le rilevazioni ottenute dovranno essere più recenti. Non può essere usato insieme a `olderThan`;
 - `{maxNumDetections}` è il numero massimo di rilevazioni da ottenere;
 - se sia `{olderThan}` che `{newerThan}` non sono specificati allora vengono restituite le ultime rilevazioni.
- **metodo HTTP:** GET;
- **body:** vuoto.

In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 200;
- **body:** formato JSON, un oggetto con i seguenti campi:
 - "detections": una lista di oggetti rappresentanti i valori rilevati e con i seguenti campi:
 - * "creationTime": un numero intero rappresentante l'istante della rilevazione come millisecondi trascorsi dall'*UNIX epoch* secondo il fuso orario UTC;
 - * "value": un numero a virgola mobile rappresentante il valore rilevato;
 - * "outlier": un booleano indicante se la rilevazione è anomala o meno.
 - "nextOld": un numero intero o valore nullo. Quando ha valore rappresenta un istante di tempo in millisecondi trascorsi dall'*UNIX epoch*. In un'altra richiesta può essere usato al posto di `olderThan`. Se non esistono rilevazioni precedenti assume valore null;
 - "nextNew": un numero intero. Quando ha valore rappresenta un istante di tempo in millisecondi trascorsi dall'*UNIX epoch*. In un'altra richiesta può essere usato al posto di `newerThan`.

In caso di esito negativo, viene restituito:

- **codice di stato HTTP:** 404;
- **body:** { "errorCode": "characteristicNotFound" }

Esso soddisfa il requisito del front-end UI: RIF6.

3.4.2.1.8 Limiti e media di una caratteristica

Questo endpoint REST permette al front-end UI di ottenere limiti di riferimento, media e deviazione standard relativi a una caratteristica per un utente. Consiste in:

- **endpoint:** /devices/{deviceId}/characteristics/{characteristicId}/limits, dove:
 - {deviceId} è l'identificativo della macchina;
 - {characteristicId} è l'identificativo della caratteristica.
- **metodo HTTP:** GET;
- **body:** vuoto.

In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 200;
- **body:** formato JSON, un oggetto con i seguenti campi:
 - "lowerLimit": un numero a virgola mobile rappresentante il limite di riferimento inferiore;
 - "upperLimit": un numero a virgola mobile rappresentante il limite di riferimento superiore;
 - "mean": un numero a virgola mobile rappresentante la media.

In caso di esito negativo, viene restituito:

- **codice di stato HTTP:** 404;
- **body:** { "errorCode": "characteristicNotFound" }

Esso soddisfa il requisito del front-end UI: RIF7.

3.4.2.1.9 Lista macchine

Questo endpoint REST permette al front-end UI di ottenere la lista di tutte le macchine per un amministratore. Consiste in:

- **endpoint:** /admin/devices;
- **metodo HTTP:** GET;
- **body:** vuoto.

Viene restituito:

- **codice di stato HTTP:** 200;
- **body:** formato JSON, una lista di oggetti con i seguenti campi:
 - "id": un numero intero rappresentante l'identificativo della macchina;
 - "name": una stringa rappresentante il nome della macchina;
 - "archived": un booleano indicante se la macchina è archiviata o meno;
 - "deactivated": un booleano indicante se la macchina è disattivata o meno.

Esso soddisfa il requisito del front-end UI: RIF8.

3.4.2.1.10 Modifica stato archiviazione macchina

Questo endpoint REST permette al front-end UI di archiviare o ripristinare una macchina scelta da un amministratore. Consiste in:

- **endpoint:** /admin/devices/{deviceId}/archived, dove:
 - {deviceId} è l'identificativo della macchina da archiviare o ripristinare.
- **metodo HTTP:** PUT;
- **body:**
 - **true**, se la macchina deve essere archiviata;
 - **false**, se la macchina deve essere ripristinata.

In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 204;
- **body:** vuoto.

Se la macchina da archiviare o ripristinare non esiste, viene restituito:

- **codice di stato HTTP:** 404;
- **body:** { "errorCode": "deviceNotFound" }.

Esso soddisfa i requisiti del front-end UI: RIF9, RIF10.

3.4.2.1.11 Modifica stato attivazione macchina

Questo endpoint REST permette al front-end UI di disattivare o riattivare una macchina scelta da un amministratore. Consiste in:

- **endpoint:** /admin/devices/{deviceId}/deactivated, dove:
 - {deviceId} è l'identificativo della macchina da disattivare o riattivare.
- **metodo HTTP:** PUT;
- **body:**
 - **true**, se la macchina deve essere disattivata;
 - **false**, se la macchina deve essere attivata.

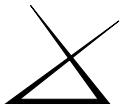
In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 204;
- **body:** vuoto.

Se la macchina da disattivare o attivare non esiste, viene restituito:

- **codice di stato HTTP:** 404;
- **body:** { "errorCode": "deviceNotFound" }.

Esso soddisfa i requisiti del front-end UI: RIF11, RIF12.



3.4.2.1.12 Inserimento macchina

Questo endpoint REST permette al front-end UI di inserire una nuova macchina per un amministratore. Consiste in:

- **endpoint:** /admin/devices;
- **metodo HTTP:** POST;
- **body:** formato JSON, un oggetto con i seguenti campi:
 - "name": una stringa rappresentante il nome della macchina;
 - "characteristics": una lista di oggetti rappresentanti le caratteristiche della macchina e con i seguenti campi:
 - * "name": una stringa rappresentante il nome della caratteristica;
 - * "lowerLimit": un numero a virgola mobile rappresentante il limite tecnico inferiore della caratteristica, null se non è stato impostato;
 - * "upperLimit": un numero a virgola mobile rappresentante il limite tecnico superiore della caratteristica, null se non è stato impostato;
 - * "autoAdjust": un booleano indicante se il motore di calcolo deve calcolare media e varianza delle rilevazioni. Se è false allora "lowerLimit" e "upperLimit" non possono essere null;
 - * "sampleSize": un numero intero rappresentante la grandezza del campione da considerare per il calcolo di media e varianza se "autoAdjust" è true, null altrimenti.

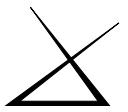
In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 200;
- **body:** formato JSON, un oggetto con i seguenti campi:
 - "id": un numero rappresentante l'identificativo della macchina appena inserita.

In caso di esito negativo, viene restituito:

- se una macchina con lo stesso nome esiste già:
 - **codice di stato HTTP:** 400;
 - **body:** { "errorCode": "duplicateDeviceName" }.
- se i requisiti relativi a autoAdjust non sono soddisfatti:
 - **codice di stato HTTP:** 400;
 - **body:** { "errorCode": "invalidValues" }.
- se la lista delle caratteristiche è vuota:
 - **codice di stato HTTP:** 400;
 - **body:** { "errorCode": "noCharacterics" }.

Esso soddisfa il requisito del front-end UI: RIF13.



3.4.2.1.13 Dettagli macchina

Questo endpoint REST permette al front-end UI di ottenere i dettagli di una macchina per un amministratore. Consiste in:

- **endpoint:** /admin/devices/{deviceId}, dove:
 - {deviceId} è l'identificativo della macchina.
- **metodo HTTP:** GET;
- **body:** vuoto.

In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 200;
- **body:** formato JSON, un oggetto con i seguenti campi:
 - "id": un numero intero identificante la macchina;
 - "name": una stringa rappresentante il nome della macchina;
 - "archived": un booleano rappresentante se la macchina è archiviata o meno;
 - "deactivated": un booleano rappresentante se la macchina è disattivata o meno;
 - "accessToken": il token di accesso per l'API rilevazioni.

Se la macchina con quell'identificativo non esiste, viene restituito:

- **codice di stato HTTP:** 404;
- **body:** { "errorCode": "deviceNotFound" }.

Esso soddisfa il requisito del front-end UI: RIF14.

3.4.2.1.14 Modifica macchina

Questo endpoint REST permette al front-end UI di modificare una macchina per un amministratore. Consiste in:

- **endpoint:** /admin/devices/{deviceId}/name, dove:
 - {deviceId} è l'identificativo della macchina.
- **metodo HTTP:** PUT;
- **body:** una stringa rappresentante il nuovo nome della macchina.

In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 204;
- **body:** vuoto.

Se una macchina con quell'identificativo non esiste, viene restituito:

- **codice di stato HTTP:** 404;
- **body:** { "errorCode": "deviceNotFound" }.
- se una macchina con lo stesso nome esiste già:
 - **codice di stato HTTP:** 400;
 - **body:** { "errorCode": "duplicateDeviceName" }.

Esso soddisfa il requisito del front-end UI: RIF15.

3.4.2.1.15 Lista delle caratteristiche di una macchina

Questo endpoint REST permette al front-end UI di ottenere le caratteristiche di una macchina per un amministratore. Consiste in:

- **endpoint:** /admin/devices/{deviceId}/characteristics, dove:
 - {deviceId} è l'identificativo della macchina.
- **metodo HTTP:** GET;
- **body:** vuoto.

In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 200;
- **body:** formato JSON, una lista di oggetti con i seguenti campi:
 - "id": un numero intero identificante la caratteristica per quella macchina;
 - "name": una stringa rappresentante la caratteristica;
 - "archived": un booleano indicante se la caratteristica è archiviata o meno.

Se non esiste una macchina con quell'identificativo, viene restituito:

- **codice di stato HTTP:** 404;
- **body:** { "errorCode": "deviceNotFound" }.

Esso soddisfa il requisito del front-end UI: RIF16.

3.4.2.1.16 Modifica stato archiviazione caratteristica

Questo endpoint REST permette al front-end UI di archiviare o ripristinare una caratteristica scelta da un amministratore. Consiste in:

- **endpoint:** /admin/devices/{deviceId}/characteristics/{characteristicId}/archived, dove:
 - {deviceId} è l'identificativo della macchina;
 - {characteristicId} è l'identificativo della caratteristica.
- **metodo HTTP:** PUT;
- **body:**
 - true, se la caratteristica deve essere archiviata;
 - false, se la caratteristica deve essere ripristinata.

In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 204;
- **body:** vuoto.

Se non esiste una caratteristica con quell'identificativo, viene restituito:

- **codice di stato HTTP:** 404;
- **body:** { "errorCode": "characteristicNotFound" }.

Esso soddisfa i requisiti del front-end UI: RIF17, RIF18.

3.4.2.1.17 Inserimento caratteristica

Questo endpoint REST permette al front-end UI di aggiungere una caratteristica a una macchina per un amministratore. Consiste in:

- **endpoint:** `/admin/devices/{deviceId}/characteristics`, dove:
 - `{deviceId}` è l'identificativo della macchina.
- **metodo HTTP:** POST;
- **body:** formato JSON, un oggetto con i seguenti campi:
 - `"name"`: una stringa rappresentante il nome della caratteristica;
 - `"lowerLimit"`: un numero a virgola mobile rappresentante il limite tecnico inferiore della caratteristica o `null` se non è stato impostato;
 - `"upperLimit"`: un numero a virgola mobile rappresentante il limite tecnico superiore della caratteristica o `null` se non è stato impostato;
 - `"autoAdjust"`: un booleano indicante se il motore di calcolo dovrà calcolare media e varianza delle rilevazioni. Se è `false` allora `lowerLimit` e `upperLimit` non possono essere `null`;
 - `"sampleSize"`: un numero intero rappresentante la grandezza del campione da considerare per il calcolo di media e varianza se `autoAdjust` è `true`, `null` altrimenti.

In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 200;
- **body:** formato JSON, un oggetto contenente i seguenti campi:
 - `id`: l'identificativo della caratteristica creata.

In caso di esito negativo, viene restituito:

- se esiste già una caratteristica associata alla macchina con il nome scelto:
 - **codice di stato HTTP:** 400;
 - **body:** { `"errorCode": "duplicateCharacteristicName"` }.
- se i requisiti relativi a `autoAdjust` non sono soddisfatti:
 - **codice di stato HTTP:** 400;
 - **body:** { `"errorCode": "invalidValues"` }.
- se non esiste una macchina con quell'identificativo:
 - **codice di stato HTTP:** 404;
 - **body:** { `"errorCode": "deviceNotFound"` }.

Esso soddisfa il requisito del front-end UI: RIF19.

3.4.2.1.18 Modifica caratteristica

Questo endpoint REST permette al front-end UI di modificare la caratteristica di una macchina per un amministratore. Consiste in:

- **endpoint:** `/admin/devices/{deviceId}/characteristics/{characteristicId}`, dove:
 - `{deviceId}` è l'identificativo della macchina;
 - `{characteristicId}` è l'identificativo della caratteristica da modificare.
- **metodo HTTP:** PUT;
- **body:** formato JSON, un oggetto con i seguenti campi:
 - `"name"`: una stringa rappresentante il nome della caratteristica;
 - `"lowerLimit"`: un numero a virgola mobile rappresentante il limite tecnico inferiore della caratteristica o `null` se non è stato impostato;
 - `"upperLimit"`: un numero a virgola mobile rappresentante il limite tecnico superiore della caratteristica o `null` se non è stato impostato;
 - `"autoAdjust"`: un booleano indicante se il motore di calcolo dovrà calcolare media e varianza delle rilevazioni. Se è `false` allora `lowerLimit` e `upperLimit` non possono essere `null`;
 - `"sampleSize"`: un numero intero rappresentante la grandezza del campione da considerare per il calcolo di media e varianza se `autoAdjust` è `true`, `null` altrimenti.

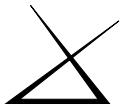
In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 204;
- **body:** vuoto.

In caso di esito negativo, viene restituito:

- se esiste già una caratteristica associata alla macchina con il nome scelto:
 - **codice di stato HTTP:** 400;
 - **body:** { `"errorCode": "duplicateCharacteristicName"` }.
- se i requisiti relativi a `autoAdjust` non sono soddisfatti:
 - **codice di stato HTTP:** 400;
 - **body:** { `"errorCode": "invalidValues"` }.
- se non esiste una macchina con quell'identificativo:
 - **codice di stato HTTP:** 404;
 - **body:** { `"errorCode": "characteristicNotFound"` }.

Esso soddisfa il requisito del front-end UI: RIF20.



3.4.2.1.19 Lista utenti

Questo endpoint REST permette al front-end UI di ottenere la lista degli utenti per un amministratore. Consiste in:

- **endpoint:** /admin/accounts;
- **metodo HTTP:** GET;
- **body:** vuoto.

Viene restituito:

- **codice di stato HTTP:** 200;
- **body:** formato JSON, una lista di oggetti con i seguenti campi:
 - "username": una stringa rappresentante il nome di un utente;
 - "administrator": un booleano indicante se l'utente è un amministratore o meno;
 - "archived": un booleano indicante se l'utente è archiviato o meno.

Esso soddisfa il requisito del front-end UI: RIF21.

3.4.2.1.20 Modifica stato archiviazione utente

Questo endpoint REST permette al front-end UI di archiviare o ripristinare un utente scelto da un amministratore. Consiste in:

- **endpoint:** /admin/accounts/{username}/archived, dove:
 - {username} è il nome dell'utente da archiviare o ripristinare.
- **metodo HTTP:** PUT;
- **body:**
 - **true**, se si vuole archiviare l'utente;
 - **false**, se si vuole ripristinare l'utente.

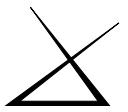
In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 204;
- **body:** vuoto.

Se l'utente non esiste, viene restituito:

- **codice di stato HTTP:** 404;
- **body:** { "errorCode": "accountNotFound" }.

Esso soddisfa i requisiti del front-end UI: RIF22, RIF23.



3.4.2.1.21 Inserimento utente

Questo endpoint REST permette al front-end UI di inserire le informazioni di un nuovo utente per un amministratore. Consiste in:

- **endpoint:** /admin/accounts;
- **metodo HTTP:** POST;
- **body:** formato JSON, un oggetto con i seguenti campi:
 - "username": una stringa rappresentante il nome del nuovo utente;
 - "password": una stringa rappresentante la password del nuovo utente;
 - "administrator": un booleano indicante se il nuovo utente è un amministratore o meno.

In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 200;
- **body:** formato JSON, un oggetto con i seguenti campi:
 - **username:** il nome del nuovo utente.

In caso di esito negativo, viene restituito:

- se l'utente esiste già:
 - **codice di stato HTTP:** 400;
 - **body:** { "errorCode": "duplicateUsername" }.
- se la password non rispetta i requisiti:
 - **codice di stato HTTP:** 400;
 - **body:** { "errorCode": "invalidPassword" }.

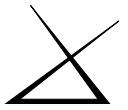
Esso soddisfa il requisito del front-end UI: RIF24.

3.4.2.1.22 Modifica utente

Questo endpoint REST permette al front-end UI di modificare un utente per un amministratore. Consiste in:

- **endpoint:** /admin/accounts/{username}, dove:
 - {username} è il nome utente dell'utente che si vuole modificare.
- **metodo HTTP:** PUT;
- **body:** formato JSON, un oggetto con i seguenti campi:
 - **password:** una stringa rappresentante la nuova password da assegnare all'utente, oppure null se si vuole mantenere la password corrente;
 - **administrator**, un booleano che vale:
 - * **true**, se si vuole rendere l'utente un amministratore;
 - * **false**, se si vuole rendere l'utente non amministratore.

In caso di esito positivo, viene restituito:



- codice di stato HTTP: 204;
- body: vuoto.

Se la password non rispetta i requisiti, viene restituito:

- codice di stato HTTP: 400;
- body: { "errorCode": "invalidPassword" }.

Esso soddisfa il requisito del front-end UI: RIF25.

3.4.2.2 Requisiti per i livelli inferiori

Tabella 3: Requisiti derivanti dal back-end UI.

Codice	Descrizione	Fonte
RIB26	<p>Il back-end deve poter persistere un'entità rappresentante un utente. Richiede di tenere traccia di:</p> <ul style="list-style-type: none">• nome utente;• hash della password;• se ha privilegi di amministratore o meno;• se è archiviato o meno.	§3.4.2.1.1 §3.4.2.1.2 §3.4.2.1.3 §3.4.2.1.4 §3.4.2.1.19 §3.4.2.1.20 §3.4.2.1.21 §3.4.2.1.22
RIB27	<p>Il back-end deve poter persistere un'entità rappresentante una macchina. Richiede di tenere traccia di:</p> <ul style="list-style-type: none">• identificativo;• nome;• se è archiviata o meno;• se è disattivata o meno.	§3.4.2.1.5 §3.4.2.1.9 §3.4.2.1.10 §3.4.2.1.11 §3.4.2.1.13 §3.4.2.1.14 §3.4.2.1.12

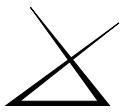
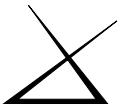


Tabella 3 (continuazione): Requisiti derivanti dal back-end UI.

Codice	Descrizione	Fonte
RIB28	<p>Il back-end deve poter persistere un'entità rappresentante una caratteristica. Richiede di tenere traccia di:</p> <ul style="list-style-type: none">• identificativo;• identificativo della macchina a cui appartiene;• nome;• limiti tecnici inferiore e superiore;• grandezza del campione di dati da utilizzare;• se l'auto-adjust è attivo o meno;• se è archiviata o meno.	§3.4.2.1.6 §3.4.2.1.8 §3.4.2.1.15 §3.4.2.1.16 §3.4.2.1.17 §3.4.2.1.18
RIB29	<p>Il back-end deve poter persistere un'entità rappresentante una rilevazione. Richiede di tenere traccia di:</p> <ul style="list-style-type: none">• istante di rilevazione;• valore;• identificativo della caratteristica a cui è associata;• se è anomala o meno.	§3.4.2.1.7



3.4.3 API rilevazioni

3.4.3.1 Definizione dell'interfaccia

3.4.3.1.1 Rilevazioni

Questo endpoint REST permette alle macchine di inviare rilevazioni al nostro prodotto. Si occupa inoltre di applicare le carte di controllo e identificare le rilevazioni anomale. Consiste in:

- **endpoint:** /detections;
- **metodo HTTP:** POST;
- **body:** formato JSON, un oggetto con i seguenti campi:
 - "apiKey": una stringa rappresentante la chiave API identificativa della macchina che invia la rilevazione;
 - "characteristic": una stringa rappresentante il nome della caratteristica rilevata;
 - "value": un numero a virgola mobile rappresentante il valore rilevato.

In caso di esito positivo, viene restituito:

- **codice di stato HTTP:** 202;
- **body:** vuoto.

In caso di esito negativo, viene restituito:

- se l'autenticazione è fallita:
 - **codice di stato HTTP:** 401;
 - **body:** { "errorCode": "notAuthenticated" }.
- se la caratteristica non esiste:
 - **codice di stato HTTP:** 404;
 - **body:** { "errorCode": "characteristicNotFound" }.
- se la macchina non accetta rilevazioni:
 - **codice di stato HTTP:** 410;
 - **body:** { "errorCode": "deviceDisabled" }.
- se la caratteristica non accetta rilevazioni:
 - **codice di stato HTTP:** 410;
 - **body:** { "errorCode": "characteristicDisabled" }.

Esso soddisfa i requisiti funzionali: ROF22, ROF22.1, ROF22.2, ROF22.3, ROF22.4, ROF22.5, ROF22.5.1, ROF22.5.2, ROF22.6, ROF22.7, ROF22.8, ROF22.9, ROF22.10, ROF22.11, ROF22.11.1.

3.4.3.2 Requisiti per i livelli inferiori

Tabella 4: Requisiti derivanti dalla API rilevazioni.

Codice	Descrizione	Fonte
RIA30	<p>L'API rilevazioni deve poter persistere i seguenti dati relativi alle macchine:</p> <ul style="list-style-type: none"> • la chiave API della macchina che invia la rilevazione; • se la macchina è abilitata o meno; • se la macchina è archiviata o meno. 	§3.4.3.1.1
RIA31	<p>L'API rilevazioni deve poter persistere i seguenti dati relativi alle caratteristiche:</p> <ul style="list-style-type: none"> • l'identificativo della caratteristica rilevata; • l'identificativo della macchina a cui appartiene; • se è archiviata o meno. 	§3.4.3.1.1
RIA32	<p>L'API rilevazioni deve poter persistere i seguenti dati relativi alle rilevazioni:</p> <ul style="list-style-type: none"> • il valore della rilevazione; • l'identificativo della caratteristica a cui è associata; • la chiave API della macchina che manda la rilevazione. 	§3.4.3.1.1

3.4.4 Database

3.4.4.1 Definizione dell'interfaccia

Il database è composto da 4 entità o tabelle:

- **device**, che rappresenta una macchina e contiene:
 - **id**: un identificativo numerico univoco;
 - **name**: il nome univoco;
 - **deactivated**: indicante se è disattivata o meno;
 - **archived**: indicante se è archiviata o meno;
 - **api_key**: la chiave API associata univoca.
- **characteristic**, che rappresenta una caratteristica e contiene:
 - **id**: un identificativo numerico univoco all'interno della sua macchina;
 - **device_id**: l'identificativo della macchina a cui è associato;
 - **name**: il nome, univoco all'interno della sua macchina;
 - **lower_limit**: il limite tecnico inferiore, se presente;

- `upper_limit`: il limite tecnico superiore, se presente;
 - `auto_adjust`: se l’auto-adjust è abilitato;
 - `sample_size`: la grandezza del campione dell’auto-adjust, se è abilitato;
 - `archived`: indicante se è archiviata o meno.
- `detection`, che rappresenta una rilevazione e contiene:
 - `creation_time`: l’istante di creazione, univoco tra le rilevazioni della stessa caratteristica;
 - `characteristic_id`: l’identificativo della caratteristica associata;
 - `device_id`: l’identificativo della macchina associata;
 - `value`: il valore misurato;
 - `outliver`: indicante se è anomala o meno.
 - `user`, che rappresenta un utente e contiene:
 - `username`: il nome univoco dell’utente;
 - `hashed_password`: l’hash della password;
 - `administrator`: indicante se è amministratore o meno;
 - `archived`: indicante se è archiviato o meno.

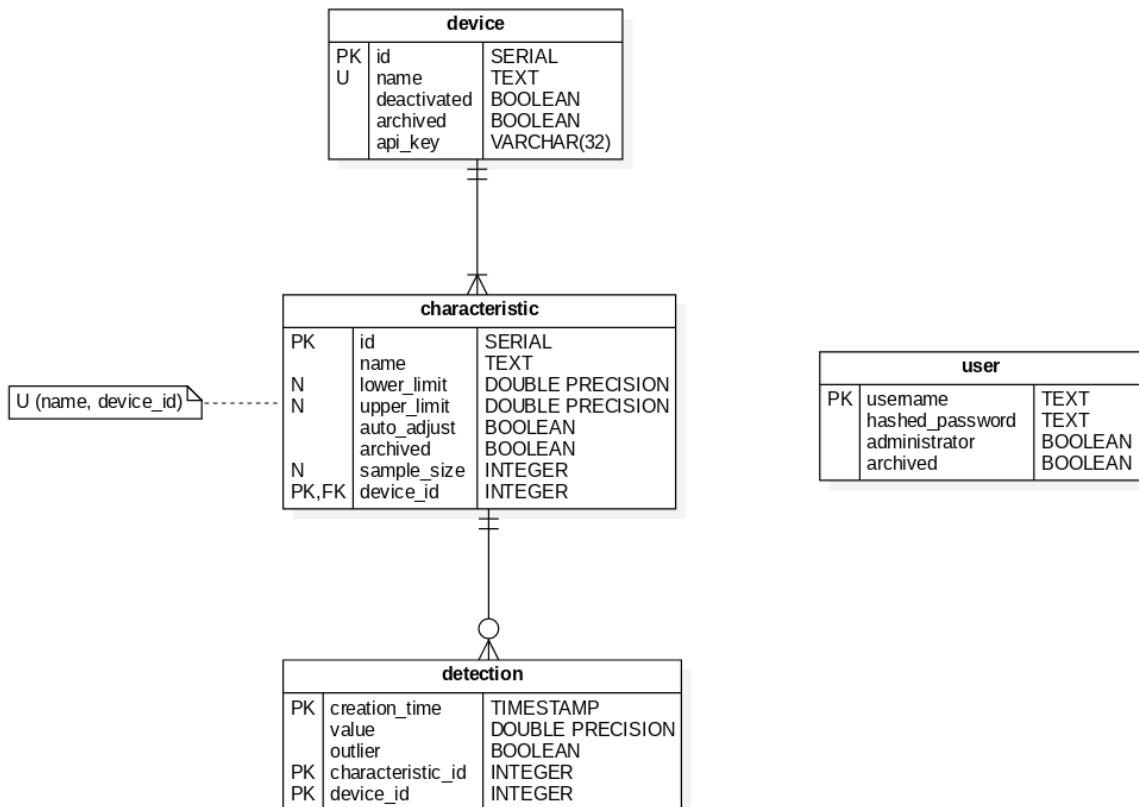
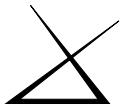


Figura 2: Diagramma ER della base di dati.



```
1 CREATE TABLE device
2 (
3     id SERIAL PRIMARY KEY,
4     name TEXT NOT NULL UNIQUE,
5     deactivated BOOLEAN NOT NULL,
6     archived BOOLEAN NOT NULL,
7     api_key VARCHAR(32) NOT NULL UNIQUE
8 );
9
10 CREATE TABLE characteristic
11 (
12     id SERIAL,
13     name TEXT NOT NULL,
14     archived BOOLEAN NOT NULL,
15     lower_limit DOUBLE PRECISION,
16     upper_limit DOUBLE PRECISION,
17     auto_adjust BOOLEAN NOT NULL,
18     sample_size INTEGER,
19     device_id INTEGER NOT NULL REFERENCES device(id),
20     PRIMARY KEY(device_id, id),
21     UNIQUE(name, device_id)
22 );
23
24 CREATE TABLE detection
25 (
26     creation_time TIMESTAMP,
27     value DOUBLE PRECISION NOT NULL,
28     outlier BOOLEAN NOT NULL,
29     characteristic_id INTEGER NOT NULL,
30     device_id INTEGER NOT NULL,
31     PRIMARY KEY(device_id, characteristic_id, creation_time),
32     FOREIGN KEY(device_id, characteristic_id)
33         REFERENCES characteristic(device_id, id)
34 );
35
36 CREATE TABLE account
37 (
38     username TEXT PRIMARY KEY,
39     hashed_password TEXT NOT NULL,
40     administrator BOOLEAN NOT NULL,
41     archived BOOLEAN NOT NULL
42 );
```

Listing 1: Script di creazione della base di dati

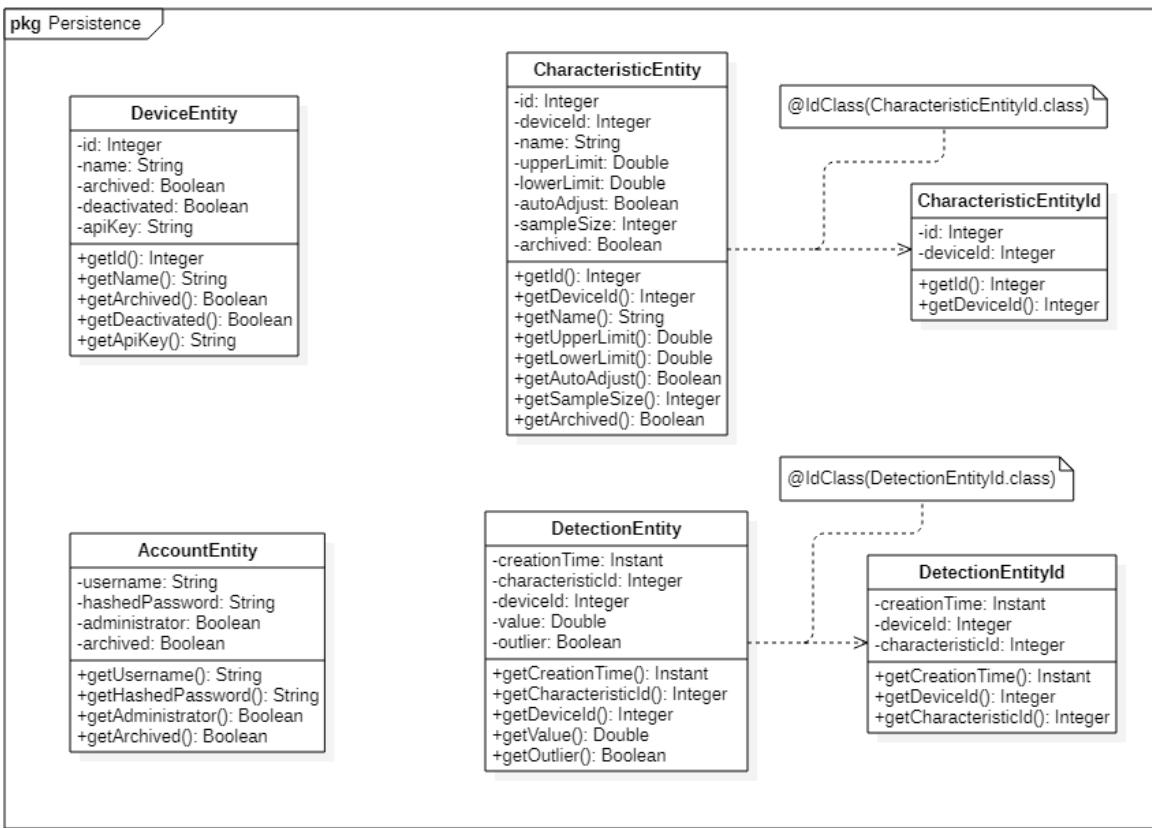


Figura 3: Diagramma delle classi della libreria di persistenza.

3.4.5 Libreria di persistenza

La libreria di persistenza raccoglie le classi che modellano le entità della base di dati. Essa è condivisa tra il back-end UI e l'API rilevazioni.

In aggiunta agli elementi descritti in questa sezione e a quelli impliciti secondo le *Norme di Progetto v2.0*, sono impliciti anche:

- un costruttore senza argomenti e protetto, utilizzabile solo da Spring;
- un getter per ogni campo.

DeviceEntity

Questa classe rappresenta una macchina salvata nella banca dati.

Annotazioni

- `@Entity`;
- `@Table(name = "device")`.

Campi privati

- `@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "id", nullable = false)
Integer id:` l'identificativo della macchina;
- `@Column(name = "name", nullable = false)
String name:` il nome della macchina;
- `@Column(name = "archived", nullable = false)
Boolean deactivated:` lo stato di attivazione della macchina;
- `@Column(name = "deactivated", nullable = false)
Boolean archived:` lo stato di archiviazione della macchina;
- `@Column(name = "api_key", nullable = false)
String apiKey:` la chiave della API rilevazioni.

Costruttori

- `DeviceEntity(
 String name: il nome della macchina;
 Boolean deactivated: lo stato di attivazione della macchina;
 Boolean archived: lo stato di archiviazione della macchina;
 String apiKey: la chiave della API rilevazioni.
)`
- Questo costruttore crea una rilevazione senza il campo *id*.

CharacteristicEntity

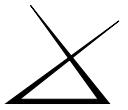
Questa classe rappresenta una caratteristica salvata nella banca dati.

Annotazioni

- `@Entity;`
- `@Table(name = "characteristic");`
- `@IdClass(CharacteristicEntityId.class).`

Campi privati

- `@GeneratedValue(strategy = GenerationType.TABLE)
@Column(name = "id", nullable = false)
@Id
Integer id:` l'identificativo della caratteristica all'interno della macchina;
- `@Column(name = "device_id", nullable = false)
@Id
Integer deviceId:` l'identificativo della macchina a cui la caratteristica appartiene;
- `@Column(name = "name", nullable = false)
String name:` il nome della caratteristica;



- `@Column(name = "upper_limit", nullable = true)`
`Double upperLimit: il limite tecnico superiore;`
- `@Column(name = "lower_limit", nullable = true)`
`Double lowerLimit: il limite tecnico inferiore;`
- `@Column(name = "auto_adjust", nullable = false)`
`Boolean autoAdjust: true se l'autoadjust è attivo; false altrimenti;`
- `@Column(name = "sample_size", nullable = true)`
`Integer sampleSize: la grandezza del campione necessario a calcolare la media e la varianza;`
- `@Column(name = "archived", nullable = false)`
`Boolean archived: lo stato di archiviazione della caratteristica.`

Costruttori

- `CharacteristicEntity(`
`Integer deviceId: l'identificativo della macchina a cui la caratteristica appartiene;`
`String name: il nome della caratteristica;`
`Double upperLimit: il limite tecnico superiore;`
`Double lowerLimit: il limite tecnico inferiore;`
`Boolean autoAdjust: true se l'autoadjust è attivo; false altrimenti;`
`Integer sampleSize: la grandezza del campione necessario a calcolare la media e la varianza;`
`Boolean archived: lo stato di archiviazione della caratteristica.`
`)`
Questo costruttore crea una caratteristica senza il campo *id*.

CharacteristicEntityId

Questa classe rappresenta l'identificativo di una caratteristica salvata nella banca dati.

Interfacce implementate

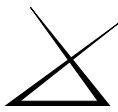
- `java.io.Serializable.`

Campi privati

- `Integer id: l'identificativo della caratteristica all'interno della macchina;`
- `Integer deviceId: l'identificativo della macchina a cui la caratteristica appartiene.`

Costruttori

- `CharacteristicEntityId(`
`Integer id: l'identificativo della caratteristica all'interno della macchina;`
`Integer deviceId: l'identificativo della macchina a cui la caratteristica appartiene.`
`)`
Questo costruttore crea la chiave di una caratteristica senza il campo *id*.



DetectionEntity

Questa classe rappresenta una rilevazione nella banca dati.

Annotazioni

- `@Entity;`
- `@Table(name = "detection");`
- `@IdClass(DetectionEntityId.class).`

Campi privati

- `@Column(name = "creation_time", nullable = false)`
`@Id`
`Instant creationTime`: l'istante della rilevazione, rappresentato dai millisecondi trascorsi dallo UNIX epoch;
- `@Column(name = "characteristic_id", nullable = false)`
`@Id`
`Integer characteristicId`: l'identificativo della caratteristica all'interno della macchina a cui la rilevazione appartiene;
- `@Column(name = "device_id", nullable = false)`
`@Id`
`Integer deviceId`: l'identificativo della macchina a cui la caratteristica appartiene;
- `@Column(name = "value", nullable = false)`
`Double value`: il valore rilevato;
- `@Column(name = "outlier", nullable = false)`
`Boolean outlier`: `true` se la rilevazione è anomala; `false` altrimenti.

DetectionEntityId

Questa classe rappresenta l'identificativo di una rilevazione salvata nella banca dati.

Interfacce implementate

- `java.io.Serializable`.

Campi privati

- `Instant creationTime`: l'istante della rilevazione, rappresentato dai millisecondi trascorsi dallo UNIX epoch;
- `Integer characteristicId`: l'identificativo della caratteristica all'interno della macchina a cui la rilevazione appartiene;
- `Integer deviceId`: l'identificativo della macchina a cui la caratteristica appartiene.

AccountEntity

Questa classe rappresenta un utente salvato nella banca dati.

Annotazioni

- `@Entity;`
- `@Table(name = "account");`
- `@IdClass(DetectionEntityId.class).`

Campi privati

- `@Column(name = "username", nullable = false)`
`@Id`
`String username:` l'username dell'utente;
- `@Column(name = "hashed_password", nullable = false)`
`String hashedPassword:` un hash della password dell'utente;
- `@Column(name = "administrator", nullable = false)`
`Boolean administrator:` i permessi dell'utente;
- `@Column(name = "archived", nullable = false)`
`Boolean archived:` lo stato di archiviazione dell'utente.

3.4.6 Diagrammi di sequenza

In questa sezione viene descritta l'interazione tipica fra le componenti del sistema in esecuzione. Viene fornito un esempio per ognuno dei due sottosistemi, ovvero l'interfaccia utente e l'API rilevazioni.

3.4.6.1 Elenco delle macchine

Quanto segue è un esempio significativo che espone l'interazione di base tra le componenti software del sistema. Come si evince dal diagramma in figura 4:

1. l'utente richiede al front-end UI di visualizzare le rilevazioni;
2. per permettere all'utente di selezionare la macchina da prendere in esame, il front-end UI invia una richiesta per l'elenco delle macchine al back-end UI;
3. alla richiesta fatta al back-end UI consegue una interrogazione allo strato di persistenza;
4. lo strato di persistenza risponde con un elenco di elementi, ognuno dei quali rappresentante una macchina;
5. l'elenco viene convertito dal back-end UI e inviato al front-end UI;
6. infine, il front-end UI mostra la lista delle macchine all'utente.

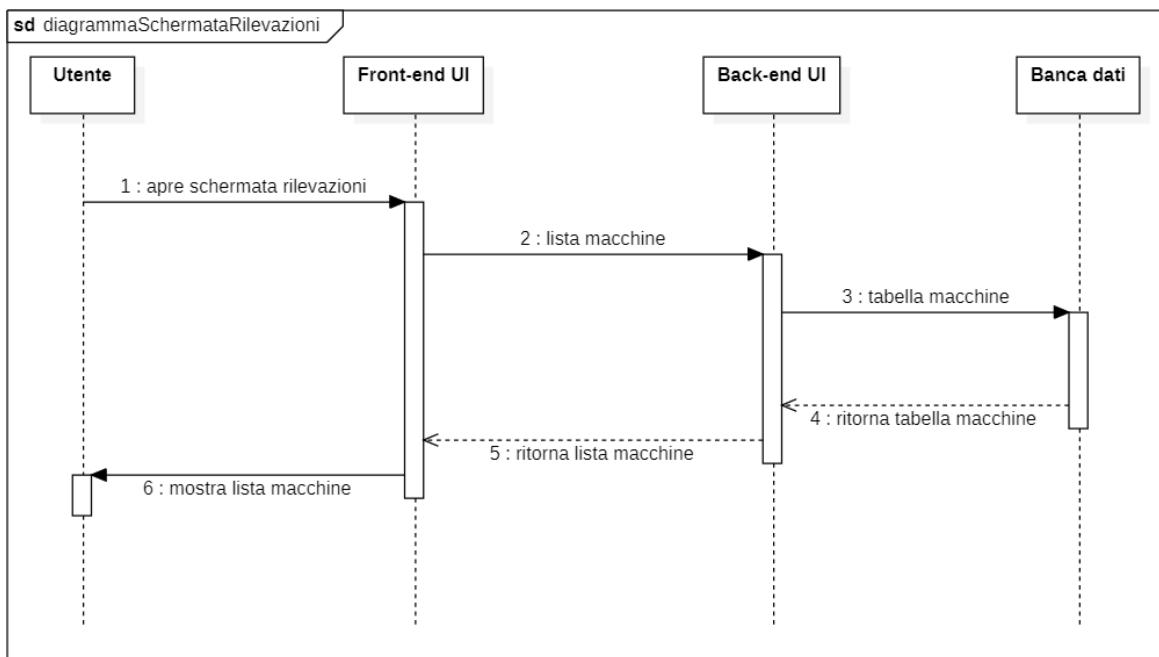


Figura 4: Diagramma di sequenza della richiesta dell'elenco delle macchine.

L'elemento chiave di queste interazioni è che sono bloccanti: l'utente deve aspettare che la richiesta arrivi fino allo strato di persistenza e che poi le informazioni attraversino tutti gli strati a ritroso. Tutte le interazioni dell'utente con la UI seguono questa struttura.

3.4.6.2 Invio delle rilevazioni

Quanto segue descrive la rilevazione del dato inviato da una macchina all'API rilevazioni e conseguente processamento nel motore di calcolo. Come si evince dal diagramma in figura 5:

1. la macchina invia una rilevazione al sistema;
2. l'API richiede le informazioni della macchina e della caratteristica allo strato di persistenza;
3. lo strato di persistenza restituisce i dati cercati;
4. il sistema controlla che le credenziali siano valide, che la macchina e la caratteristica esistano e che non siano archiviate o disattivate. L'API invia subito alla macchina l'esito della validazione;
5. dopo la risposta, il motore di calcolo richiede le ultime rilevazioni allo strato di persistenza;
6. lo strato di persistenza invia le ultime rilevazioni;
7. il motore di calcolo effettua i calcoli necessari per stabilire se la serie di punti sia anomala o meno e invia l'esito alla banca dati.

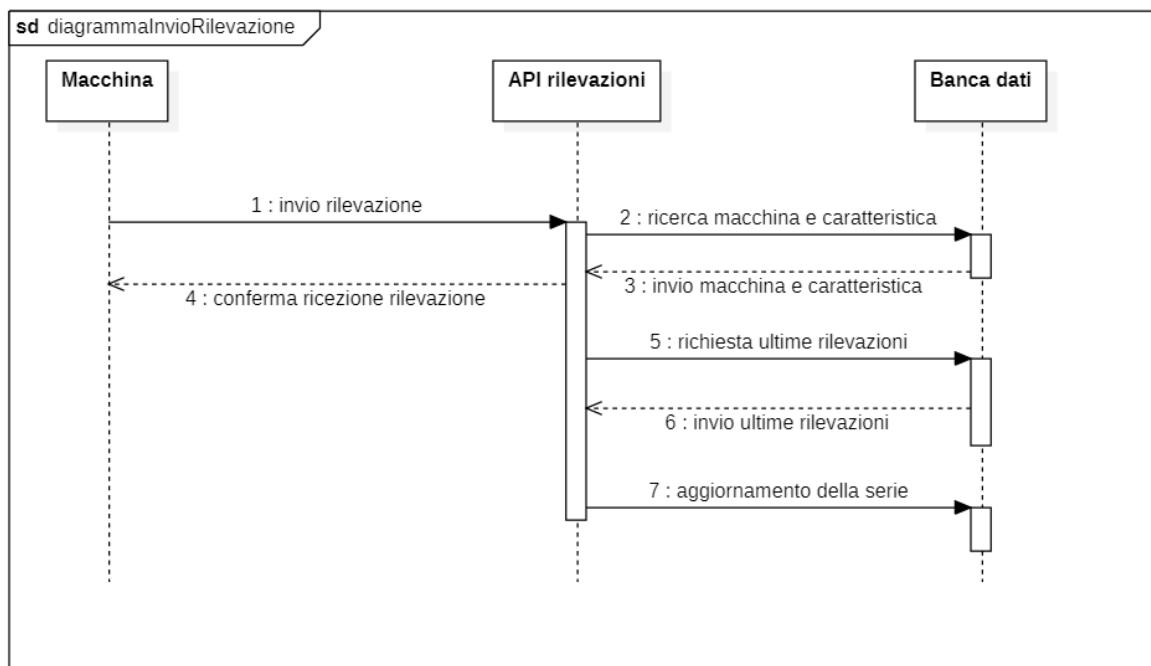
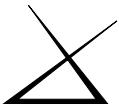


Figura 5: Diagramma di sequenza della ricezione di una rilevazione.

L'elemento chiave di queste interazioni è che l'accettazione di una rilevazione non viene bloccata dal calcolo. Il sistema di controllo di una macchina, infatti, non ha bisogno di sapere se il punto inviato costituisce un'anomalia, ma solo se la rilevazione è andata a buon fine.

4 Architettura delle componenti

Questa sezione descrive l'architettura interna di ogni componente.

Per le componenti back-end UI e API rilevazioni è implicita:

- l'esistenza di un costruttore che accetta un argomento per campo e lo assegna a tale campo, a meno che non sia definito un altro costruttore;
- l'esistenza nei record di un getter per campo, con lo stesso nome.

4.1 Front-end UI

L'interfaccia utente, di seguito “front-end UI” o anche solo “UI”, è sviluppata usando il framework Angular versione 13. La progettazione segue lo standard proposto dal framework, ovvero il pattern architettonico Model-View-ViewModel (MVVM). Essendo un progetto TypeScript, le unità di codice (`.ts`) sono organizzate in cartelle. La suddivisione delle cartelle segue questa logica:

- `app` contiene i file di $View_G$ e $ViewModel_G$, categorizzati per $component_G$, e dunque in base alle schermate di visualizzazione;
- `model` contiene i file del $Model_G$, ovvero i veri e propri file di modello e i $service_G$ che fanno parte del sistema di dependency injection di Angular. Questi file sono categorizzati per le entità REST messe a disposizione dal back-end.

Ogni component viene costituito da:

- un template in HTML5, che costituisce la View;
- una classe TypeScript, con l'annotazione `@Component` che la identifica come tale nel sistema di dependency injection di Angular. Questa classe costituisce il ViewModel;
- un file `.spec.ts`, contenente i test di unità per quel component;
- un file CSS per raccogliere gli stili specifici per il component.

Altri elementi eventuali sono:

- un file con estensione `.module.ts`, ovvero un $module_G$ che dichiara ed eventualmente esporta:
 - il component principale (con estensione `.component.ts`);
 - i component che sono raggiungibili o utilizzati dal component principale;
 - i component ai quali è associato un percorso URL comune.

I moduli completamente dedicati al $routing_G$ si chiamano `NOME-routing.module.ts`, dove `NOME` è il nome della cartella e/o del path a cui tutti i componenti dichiarati sono subordinati;

- un file con estensione `.data-source.ts`, ovvero un sorgente di dati asincroni per visualizzazioni di dati complesse. In caso si ritenga indispensabile l'utilizzo di classi o interfacce a supporto di strutture dati complesse, allora sono raggruppate in una cartella.

Ogni cartella del modello è associata a una risorsa REST del back-end e contiene:

- uno o più **servizi astratti**: classi astratte che modellano il servizio offerto dal back-end per quella risorsa;
- una **classe concreta** che implementa (secondo le regole di TypeScript le classi possono effettivamente implementare altri tipi trattandoli come interfacce) un servizio astratto, per ogni servizio astratto. Tale classe effettua, mediante un'istanza dependency injected della classe di utilità di Angular `HttpClient`, delle operazioni HTTP verso il back-end;

- i tipi, classi o interfacce, di modello utilizzati da tali servizi.

Al fine di semplificare le attività di verifica manuale delle interfacce, oltre che automatica mediante test di unità con lo strumento Karma, viene configurato il sistema di profilazione come segue:

- viene sfruttato il meccanismo *environment_G* di Angular, che permette di utilizzare una configurazione diversa in base al fatto che l'UI sia in esecuzione
 - in produzione (`environment.prod.ts`);
 - o in fase di sviluppo (`environment.ts`).
- viene definito un tipo, `Prodlytics Environment`, al quale le diverse configurazioni di ambiente devono conformarsi, in modo tale da prevenire gli errori di configurazione. A questo tipo, per ogni servizio astratto, viene aggiunto un attributo di tipo `Type<A>`, dove A è il servizio astratto. Gli oggetti di environment vengono dunque obbligati a dichiarare una classe che implementi il tipo A;
- il sistema di dependency injection di Angular viene informato dell'associazione tra i tipi astratti (richiesti dagli svariati componenti) e i tipi concreti. Questa associazione è consultabile nel file `src/app/app.module.ts`;
- di conseguenza, a runtime il sistema sa in autonomia se utilizzare una classe di produzione oppure di sviluppo.

Le classi che vengono configurate in fase di sviluppo sono raccolte nella cartella `mock`. Esse sono suddivise in cartelle, in base all'entità REST che rappresentano. In ognuna di queste cartelle viene posto un service che emula l'interazione con gli strati inferiori.

È importante notare che, da un punto di vista esclusivamente tecnico, è necessario istruire Angular con l'injection “classica” (mediante `@Injectable({providedIn: "root"})`), poiché altrimenti il meccanismo di dependency injection non condividerà il servizio mock tra i richiedenti con tipi astratti diversi.

Riassumendo, lanciando il comando `ng serve`, sarà possibile effettuare test manuali di interfaccia, effettuando tutte le operazioni su dati di prova definiti nei service mock.

4.1.1 HttpInterceptor

Questo service intercetta ogni richiesta HTTP uscente aggiungendo agli header un token di accesso.

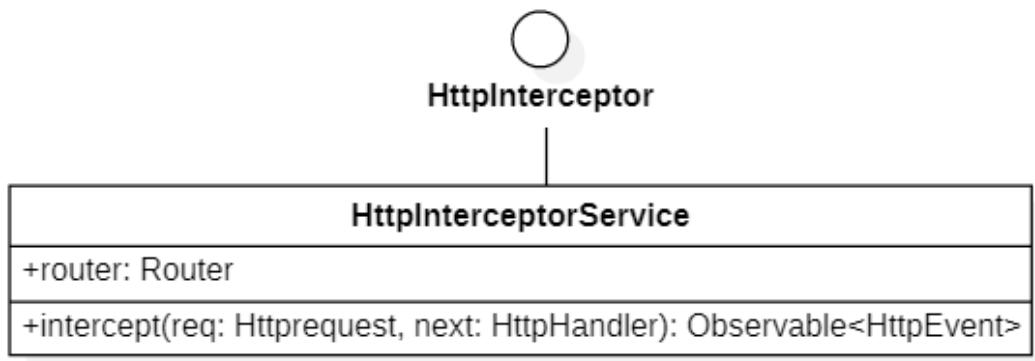


Figura 6: Diagramma delle classi dell'interceptor.

4.1.1.1 HttpInterceptorService

Annotazioni

- `@Injectable`, per poter essere identificato all'interno del sistema di dependency injection di Angular.

Campi pubblici

- `router`: Router - permette la navigazione tra differenti URL.

Metodi pubblici

- `intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent>`
Intercepta ogni chiamata HTTP uscente aggiungendo agli header un token di accesso, per poi passarla al back-end.

Interfacce implementate

- `HttpInterceptor`

4.1.2 Guards

Questi componenti impediscono a determinati utenti di raggiungere tramite URL pagine alle quali non dovrebbero avere accesso.

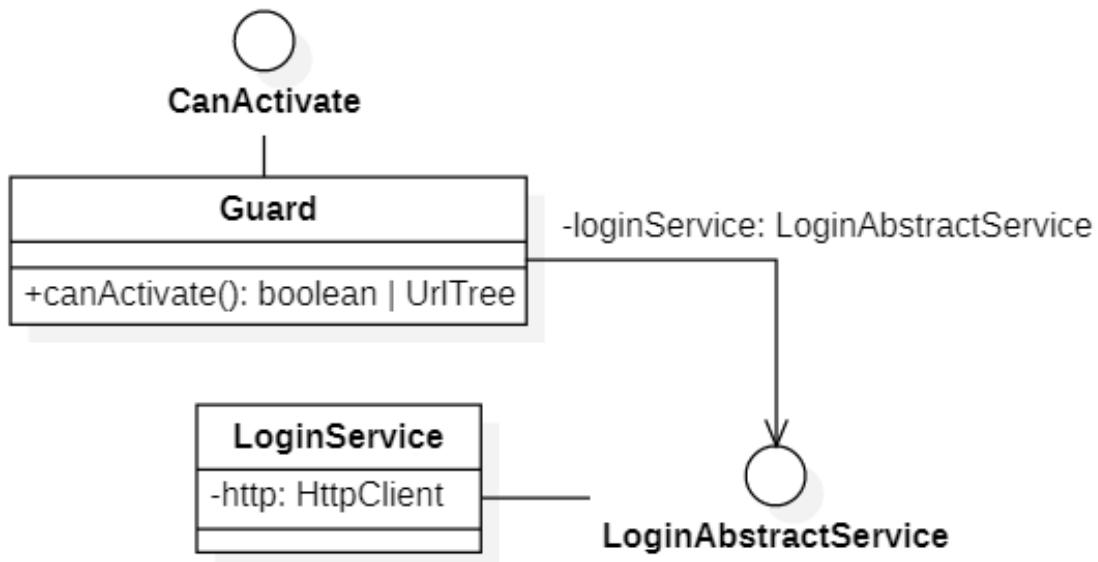
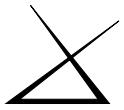


Figura 7: Diagramma delle classi di una guard.



4.1.2.1 AdminGuard

Annotazioni

- `@Injectable`, per poter essere identificato all'interno del sistema di dependency injection di Angular.

Campi privati

- `router: Router` - permette la navigazione tra differenti URL;
- `loginService: LoginAbstarctService` - permette di eseguire varie operazioni, come login e logout, e di ottenere le informazioni della sessione.

Metodi pubblici

- `canActivate(): boolean | UrlTree`
Se l'utente è un amministratore, ritorna `true`, `false` altrimenti.

Interfacce implementate

- `CanActivate`

4.1.2.2 LoginGuard

Annotazioni

- `@Injectable`, per poter essere identificato all'interno del sistema di dependency injection di Angular.

Campi privati

- `router: Router` - permette la navigazione tra differenti URL;
- `loginService: LoginAbstarctService` - permette di eseguire varie operazioni, come login e logout, e di ottenere le informazioni della sessione.

Metodi pubblici

- `canActivate(): boolean | UrlTree`
Se l'utente è autenticato, ritorna `true`, ritorna alla schermata di monitoraggio rilevazioni altrimenti.

Interfacce implementate

- `CanActivate`

4.1.2.3 AuthenticatedUserGuard

Annotazioni

- `@Injectable`, per poter essere identificato all'interno del sistema di dependency injection di Angular.

Campi privati

- `router`: `Router` - permette la navigazione tra differenti URL;
- `loginService`: `LoginAbstractService` - permette di eseguire varie operazioni, come login e logout, e di ottenere le informazioni della sessione.

Metodi pubblici

- `canActivate(): boolean | UrlTree`
Se l'utente è autenticato, ritorna `true`, porta alla schermata di login altrimenti.

Interfacce implementate

- `CanActivate`

4.1.3 Schermata di autenticazione

Questo componente implementa l'interfaccia descritta in §3.4.1.1.1.

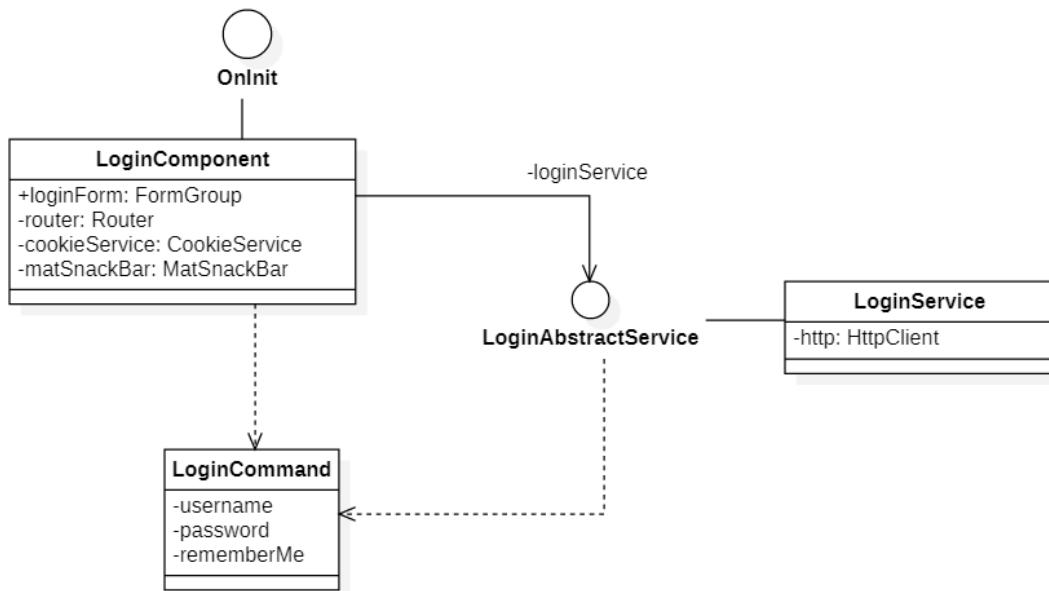
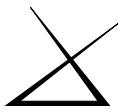


Figura 8: Diagramma delle classi della schermata di autenticazione.

4.1.3.1 LoginComponent

Annotazioni

- `@Component`, nella quale viene definita la configurazione della classe.



View

La pagina HTML5 della schermata di autenticazione comprende:

- un `input` per inserire il nome utente;
- un `input` per inserire la password;
- un `checkbox` per ricordare la sessione o meno;
- un `button` per tentare di autenticarsi;
- elementi vari di design di Angular Material.

Campi privati

- `router`: `Router` - permette la navigazione tra differenti URL;
- `loginService`: `LoginAbstractService` - permette di eseguire varie operazioni, come login e logout, e di ottenere le informazioni della sessione;
- `cookieService`: `CookieService` - permette di eseguire operazioni di base con i cookie come `get` e `set`;
- `notificationService`: `NotificationService` - fornisce un feedback grafico all'utente, come un errore o una conferma di successo di un'operazione.

Campi pubblici

- `loginForm`: `FormGroup` - un oggetto che rappresenta un reactive form di Angular. In questo oggetto vengono definiti i seguenti controlli:
 - `username`: un `FormControl` che gestisce la validazione del nome utente;
 - `password`: un `FormControl` che gestisce la validazione della password;
 - `rememberMe`: un `FormControl` che gestisce la validazione della memorizzazione della sessione.

Metodi pubblici

- `ngOnInit(): void`
Controlla se è già presente un cookie di sessione e, in caso positivo, porta alla pagina di monitoraggio rilevazioni;
- `onSubmit(): void`
Esegue un tentativo di login con le credenziali inserite dall'utente. In caso di successo l'utente viene autenticato e portato alla pagina di monitoraggio rilevazioni, altrimenti viene visualizzato un errore.

4.1.3.2 LoginAbstractService

Annotazioni

- `@Injectable`, per poter essere identificato all'interno del sistema di dependency injection di Angular.

Metodi pubblici

- `login(command: LoginCommand): Observable<>`
Permette di effettuare l'accesso dati nome utente, password e rememberMe come parametro;
- `isLoggedIn(): boolean`
Restituisce `true` se l'utente è autenticato, `false` altrimenti;
- `isAdmin(): boolean`
Restituisce `true` se l'utente è un amministratore, `false` altrimenti;
- `getUsername(): string`
Restituisce lo username dell'utente;
- `logout(): void`
Permette di effettuare il logout.

4.1.3.3 LoginService

Annotazioni

- `@Injectable`, per poter essere identificato all'interno del sistema di dependency injection di Angular.

Interfacce implementate

- `LoginAbstractService`.

Campi privati

- `http: HttpClient` - permette di effettuare richieste HTTP.

4.1.4 Barra di navigazione

Questo component implementa l'interfaccia descritta in §3.4.1.1.2.

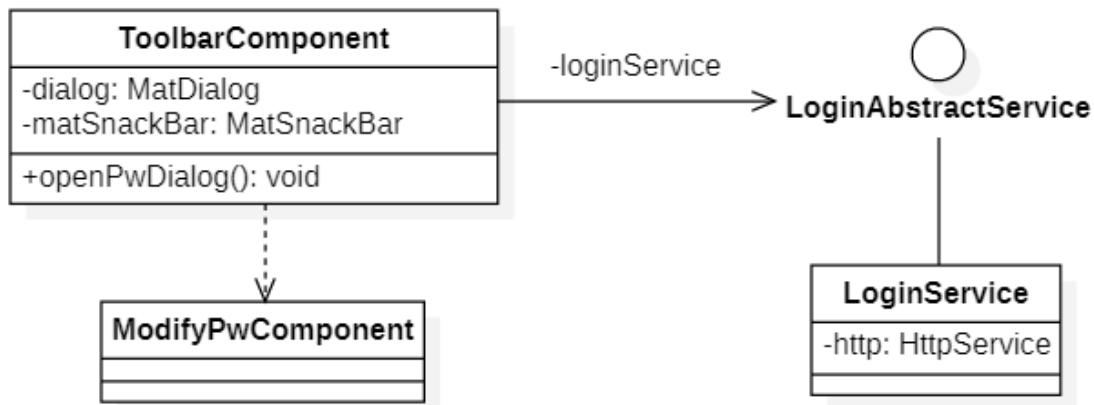
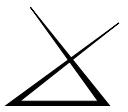


Figura 9: Diagramma delle classi della barra di navigazione.



4.1.4.1 ToolbarComponent

Annotazioni

- `@Component`, nella quale viene definita la configurazione della classe.

View

La pagina HTML5 della schermata di autenticazione comprende:

- una `mat-toolbar` che costituisce il corpo del component;
- un `button` per tornare alla pagina di monitoraggio rilevazioni;
- un `button` per visualizzare il menù utente;
- un `mat-menu` contenente quattro `mat-menu-item`, ognuno con una delle seguenti funzioni:
 - porta l'utente alla pagina di gestione macchine (visibile solo se l'utente è un amministratore);
 - porta l'utente alla pagina di gestione utenti (visibile solo se l'utente è un amministratore);
 - permette all'utente di modificare la propria password;
 - permette all'utente di effettuare il logout.
- elementi vari di design di Angular Material.

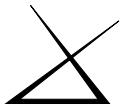
La barra di navigazione è visibile solo se l'utente ha effettuato l'accesso.

Campi privati

- `dialog`: `MatDialog` - permette l'apertura di finestre di dialogo;
- `loginService`: `LoginAbstarctService` - permette di eseguire varie operazioni, come login e logout, e di ottenere le informazioni della sessione;
- `notificationService`: `NotificationService` - permette di fornire un feedback grafico all'utente, come un errore o una conferma di successo di un'operazione;
- `router`: `Router` - permette la navigazione tra differenti URL.

Metodi pubblici

- `openPwDialog(): void`
Apri la finestra di dialogo per la modifica della password e, in base alla risposta che ottiene, conferma il successo dell'operazione oppure mostra un errore appropriato all'utente tramite `MatSnackBar`;
- `isLoggedIn(): boolean`
Restituisce `true` se l'utente è autenticato, `false` altrimenti;
- `isAdmin(): boolean`
Restituisce `true` se l'utente è un amministratore, `false` altrimenti;
- `getUsername(): string`
Restituisce lo username dell'utente;
- `logout(): void`
Esegue il logout.



4.1.5 Schermata di modifica password

Questo component implementa l'interfaccia descritta in §3.4.1.1.4.

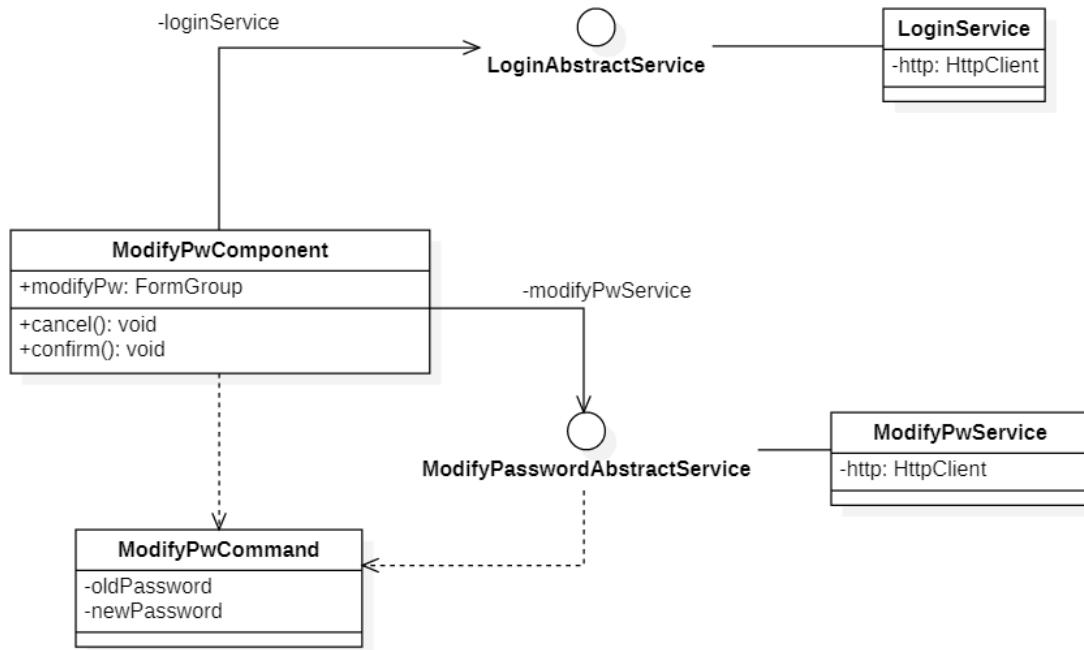


Figura 10: Diagramma delle classi della schermata di modifica password.

4.1.5.1 ModifyPwComponent

Annotazioni

- `@Component`, nella quale viene definita la configurazione della classe.

View

La pagina HTML5 dell'interfaccia contiene:

- una `mat-toolbar` contenente il titolo “Modifica password” e un `mat-button` per chiudere la finestra;
- un form con i seguenti campi:
 - password corrente, con relativi messaggi di errore;
 - nuova password, con relativi messaggi di errore;
 - conferma nuova password, con relativi messaggi di errore.
- un `mat-button` per confermare;
- un `mat-button` per annullare.

Interfacce implementate

- `OnInit`, necessaria per agganciare il life-cycle hook `ngOnInit`.

Campi privati

- `matDialogRef: MatDialogRef<ModifyPwComponent>` - si riferisce a una finestra di dialogo aperta con `MatDialog`;
- `modifyPwService: ModifyPwAbstractService` - permette di modificare la password.

Campi pubblici

- `modifyPw: FormGroup` - un oggetto che rappresenta un reactive form di Angular. In questo oggetto vengono definiti i seguenti controlli:
 - `oldPassword`: un `FormControl` che gestisce la validazione della password corrente;
 - `newPassword`: un `FormControl` che gestisce la validazione della nuova password;
 - `newPasswordRe`: un `FormControl` che gestisce la validazione della ripetizione della nuova password.

Metodi pubblici

- `checkPasswords(control: AbsractControl): validatorFn | null`
Controlla che gli elementi inseriti nell'input siano validi e restituisce gli errori appropriati;
- `confirm(): void`
Esegue un tentativo di modifica della password;
- `cancel(): void`
Chiude la finestra di dialogo.

4.1.5.2 ModifyPwAbstractService

Annotazioni

- `@Injectable`, per poter essere identificato all'interno del sistema di dependency injection di Angular.

Metodi pubblici

- `modifyPw(username: string, command: ModifyPwCommand): Observable<>`
Esegue un tentativo di modifica della password.

4.1.5.3 ModifyPwService

Annotazioni

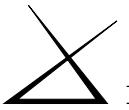
- `@Injectable`, per poter essere identificato all'interno del sistema di dependency injection di Angular.

Interfacce implementate

- `ModifyPwAbstractService`.

Campi privati

- `http: HttpClient` - permette di effettuare richieste HTTP.



4.1.6 Schermata di monitoraggio rilevazioni

4.1.6.1 SelectionComponent

Un albero che permette di selezionare caratteristiche di diverse macchine.

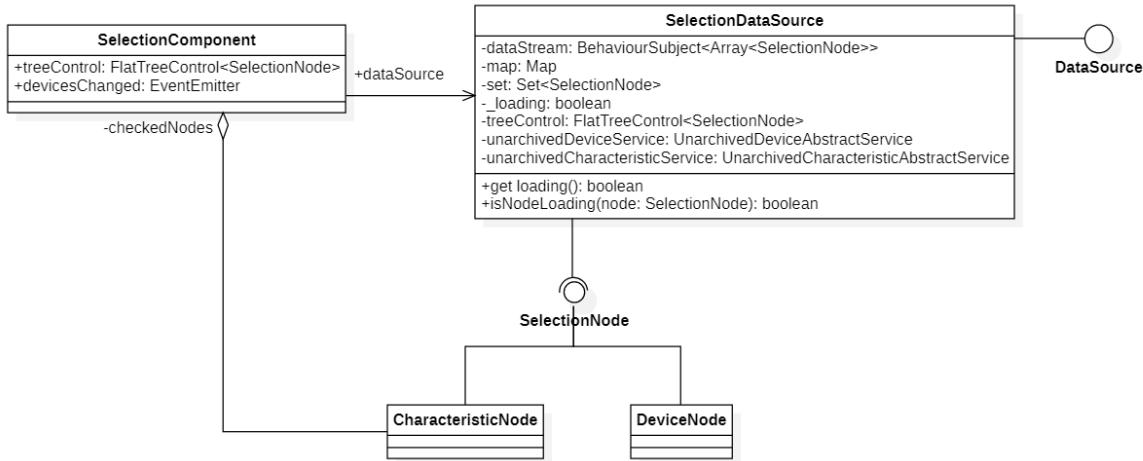


Figura 11: Diagramma delle classi dell'albero di selezione caratteristiche.

Annotazioni

- `@Component`, che configura la classe come un component di Angular.

View

La pagina HTML5 di questo component contiene:

- una `mat-sidenav` per contenere l'albero di selezione delle caratteristiche;
- un `mat-tree`, che permette di navigare tra macchine e caratteristiche:
 - i nodi sono macchine e mettono a disposizione un `button` per visualizzare o nascondere le caratteristiche;
 - le foglie sono le caratteristiche della macchina e dispongono di una casella che permette di includere o escludere la caratteristica dall'elenco dei grafici.
- un `button` che notifica il component dei grafici.

Campi pubblici

- `treeControl: FlatTreeControl<SelectionNode>` - l'oggetto che modella il controllo dell'albero;
- `dataSource: SelectionDataSource` - la sorgente dati dell'albero;
- `@Output() devicesChanged: Event Emitter<CharacteristicNode[]>` - emette le caratteristiche selezionate a ogni conferma. Può essere agganciato nei template grazie all'annotazione `@Output`.

Campi privati

- `checkedNodes: CharacteristicNode[]` - i nodi selezionati;
- `_checkedNodes: CharacteristicNode[]` - variabile che verrà passata in input a `ChartComponent` contenente i nodi da rappresentare.

Metodi pubblici

- `hasChildren(`

- `index: number`: l'indice del nodo nell'array che rappresenta l'albero;
 - `node: SelectionNode`: il nodo.

`): boolean}`

Metodo richiesto dalla `mat-table`, usato per determinare se un nodo ha o meno foglie;

- `nodeIsChecked(`

- `node: CharacteristicNode`: il nodo da controllare.

`): boolean`

Permette di determinare se il nodo passato come parametro è selezionato;

- `toggleNodeCheck(`

- `checked: boolean`: `true` se il nodo debba essere selezionato; `false`, altrimenti;
 - `node: CharacteristicNode`: il nodo da selezionare o deselectionare.

`): void`

Seleziona o deselectiona il nodo, in base allo stato dell'evento;

- `notifyChange(): void`

Copia il valore di `checkedNodes` tramite `slice()` e lo assegna a `_checkedNodes`.

4.1.6.2 SelectionDataSource

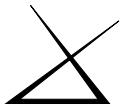
Una classe che rappresenta la sorgente dati per un albero di macchine e caratteristiche non archiviate.

Interfacce implementate

- `DataSource<SelectionNode>`, permette di poterla usare come sorgente dati asincrona.

Campi privati

- `dataStream: BehaviorSubject<SelectionNode[]>` - notifica il cambiamento dei dati;
- `cache: Map<SelectionNode, CharacteristicNode[]>` - una semplice cache per velocizzare richieste possibilmente pesanti;
- `loadingNodes: Set<SelectionNode>` - indica quali nodi siano in attesa delle foglie;
- `_loading: Boolean` - indica se l'albero stia aspettando le radici;
- `treeControl: FlatTreeControl<SelectionNode>` - il controllo dell'albero, che notifica le azioni dell'utente;



- `unarchivedDeviceService`: `UnarchivedDeviceAbstractService` - permette di cercare tra le macchine non archiviate;
- `unarchivedCharacteristicService`: `UnarchivedCharacteristicAbstractService` - permette di cercare tra le caratteristiche non archiviate.

Metodi pubblici

- `isNodeLoading()`
 - `node`: `SelectionNode`: il nodo da verificare se sia in attesa dei figli.
- `boolean`
Indica se il nodo sta aspettando le foglie.

4.1.6.3 DeviceSelection

Un'interfaccia che include i dati da mostrare in un `mat-tree`.

Campi pubblici

- `level`: `number` - il livello di profondità del nodo;
- `expandable`: `boolean` - indica se il nodo è espandibile o meno;
- `id`: `number` - l'identificativo del nodo;
- `name`: `string` - il nome da visualizzare in interfaccia.

4.1.6.4 DeviceNode

Espone i dati di una macchina in un albero.

Interfacce implementate

- `SelectionNode`, permette di utilizzare questa classe come nodo in un albero.

Costruttore

Il costruttore prende i seguenti parametri:

- `device`: `Device` - la macchina da rappresentare.

4.1.6.5 CharacteristicNode

Espone i dati di una caratteristica e della sua macchina in un albero.

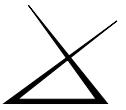
Interfacce implementate

- `SelectionNode`, permette di utilizzare questa classe come nodo in un albero.

Costruttore

Il costruttore prende i seguenti parametri:

- `device`: `SelectionNode` - il nodo padre di questa caratteristica;
- `characteristic`: `Characteristic` - la caratteristica da rappresentare.



Campi pubblici

- device: SelectionNode - un riferimento al nodo padre.

4.1.6.6 ChartComponent

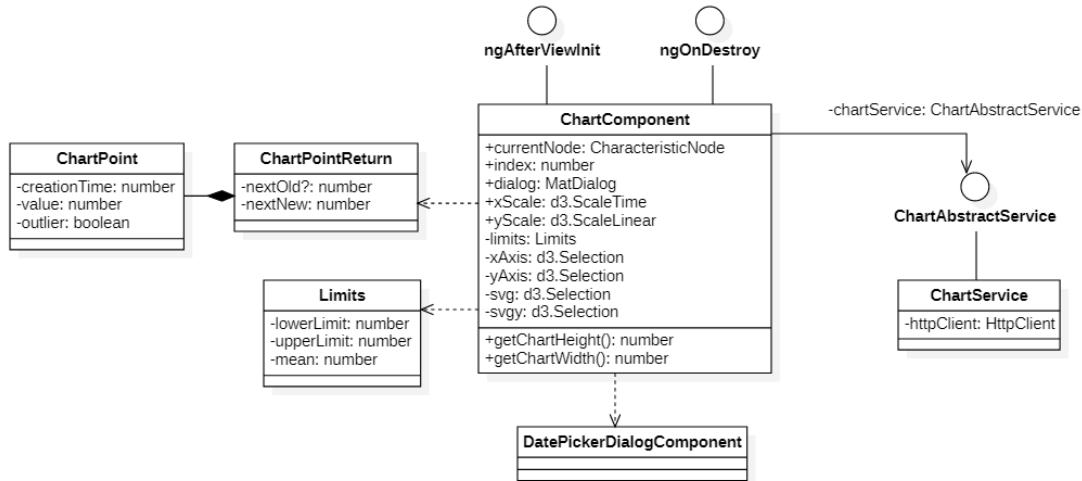


Figura 12: Diagramma delle classi del componente di visualizzazione grafici.

Annotazioni

- `@Component`, nella quale viene definita la configurazione della classe.

View

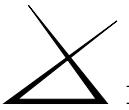
- una `mat-toolbar` con:
 - nomi della macchina e della caratteristica visualizzata;
 - un `mat-button` per inserire l’intervallo temporale entro il quale si vogliono vedere le rilevazioni;
 - un `mat-button` per ricaricare il grafico.
- due `svg`, uno per rappresentare l’asse delle ordinate, l’altro per rappresentare il resto del grafico.

Interfacce implementate

- `@AfterViewInit`, necessaria per agganciare il life-cycle hook `ngAfterViewInit`;
- `@OnDestroy`, necessaria per agganciare il life-cycle hook `ngOnDestroy`.

Campi pubblici

- `@Input() currentNode: CharacteristicNode` - un nodo dell’albero di selezione caratteristiche selezionato. È preso in input dal template;
- `@Input() index: number` - un numero intero utilizzato per dare un `id` progressivo ai vari grafici e per riferirsi a essi. È preso in input dal template.

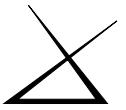


Campi privati

- `limits`: `Limits` - i valori dei limiti superiore e inferiore e della media;
- `points`: `Points[]` - i punti correntemente mostrati nel grafico;
- `updateSubscription`: `Subscription` - rappresenta la `subscription` a `ChartService`;
- `xAxis`: `d3.Selection<SVGGEElement, unknown, HTMLElement, any>` - l'asse delle ascisse del grafico;
- `yAxis`: `d3.Selection<SVGGEElement, unknown, HTMLElement, any>` - l'asse delle ordinate del grafico;
- `chartService`: `ChartAbstractService` - permette di ottenere i valori di punti, limiti e media;
- `svgy`: `d3.Selection<SVGGEElement, unknown, HTMLElement, any>` - l'svg contente l'asse delle ordinate del grafico;
- `svg`: `d3.Selection<SVGGEElement, unknown, HTMLElement, any>` - il resto del grafico;
- `xScale`: `d3.ScaleTime<number, number, never>` - scala temporale usata per l'asse delle ascisse;
- `yScale`: `d3.ScaleLinear<number, number, never>` - scala lineare usata per l'asse delle ordinate;
- `dialog`: `MatDialog` - permette l'apertura di finestre di dialogo.

Metodi pubblici

- `ngAfterViewInit(): void`
Chiama `getData()`, `createChart()` e `subscribeToUpdates` in questo ordine, dopo che la view è stata inizializzata;
- `ngOnDestroy(): void`
Rimuove la `subscription` dopo che l'istanza è stata chiusa;
- `get chartWidth(): number`
Ritorna la larghezza del grafico;
- `get chartHeight(): number`
Ritorna l'altezza del grafico;
- `openDialog(): void`
Apre `DatePickerDialogComponent`. Nel caso in cui l'utente inserisca dei dati, viene ricreato il grafico;
- `createChart(): void`
Istanzia gli elementi di base del grafico, come i limiti, gli assi e i punti;
- `getData(deviceId: number, characteristicId: number): void`
Assegna valori alle variabili `points` e `limits`;
- `drawChart(): void`
Rappresenta graficamente gli elementi inizializzati da `createChart()`;
- `subscribeToUpdates(): void`
Ogni secondo aggiorna `points` con una nuova rilevazione, contemporaneamente rimuove la più vecchia visualizzata;



- **clearChart(): void**
Rimuove tutti gli elementi del grafico e la subscription;
- **refresh()**
Oltre a chiamare il metodo `clearChart()`, ricrea il grafico.

4.1.6.7 ChartAbstractService

Annotazioni

- `@Injectable`, per poter essere identificato all'interno del sistema di dependency injection di Angular.

Metodi pubblici

- **getInitialPoints(deviceId: number, characteristicId: number): Observable<ChartPointReturn>**
Fornisce l'elenco dei punti iniziali del grafico;
- **getNextPoints(deviceId: number, characteristicId: number, latestEpoch: number): Observable<ChartPointReturn>**
Fornisce i punti più recenti di `latestEpoch`;
- **getLimits(deviceId: number, characteristicId: number): Observable<Limits>**
Fornisce media e limiti della caratteristica `characteristicId`;
- **getOldPoints(start: number, end: number, deviceId: number, characteristicId: number): Observable<ChartPointReturn>**
Fornisce l'elenco delle rilevazioni più recenti di `start` e più vecchie di `end`.

4.1.6.8 ChartService

Annotazioni

- `@Injectable`, per poter essere identificato all'interno del sistema di dependency injection di Angular.

Interfacce implementate

- `ChartAbstractService`.

Campi privati

- `httpClient: HttpClient` - permette di effettuare richieste HTTP.

4.1.6.9 DatepickerDialogComponent

Annotazioni

- `@Component`, nel quale viene definita la configurazione della classe.

View

- un `mat-form-field` in cui è possibile selezionare una data di inizio e una di fine;
- due `ngb-timepicker` in cui è possibile scegliere ore, minuti e secondi iniziali e finali;
- due `mat-button`, uno per confermare e l'altro per annullare.

Campi pubblici

- `dateForm: FormGroup` - un oggetto che rappresenta un reactive form di Angular. In questo oggetto vengono definiti i seguenti controlli:
 - `start`: un `FormControl` che gestisce la validazione della data iniziale;
 - `end`: un `FormControl` che gestisce la validazione della data finale.

Campi privati

- `matDialogRef: MatDialogRef<DatePickerDialogComponent>` - si riferisce a una finestra di dialogo aperta con `MatDialog`.

Metodi pubblici

- `confirm(): void`
Converte i dati inseriti dall'utente in formato UNIX epoch e li passa a `ChartComponent`;
- `cancel(): void`
Chiude la finestra di dialogo.

4.1.7 Schermata di gestione macchine

La pagina di gestione macchine è contenuta e gestita dal modulo `src/app/admin/devices/devices.module.ts`. Questo component implementa l'interfaccia descritta in §3.4.1.1.6.

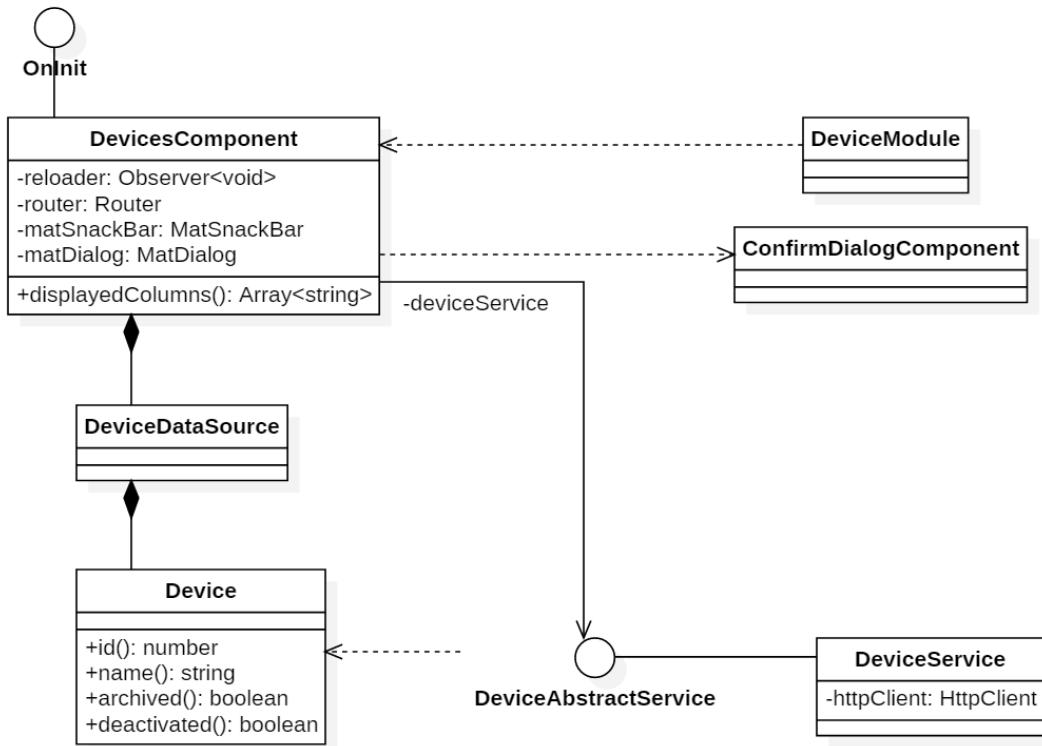
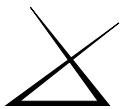


Figura 13: Diagramma delle classi della schermata di gestione macchine.



4.1.7.1 DevicesComponent

Annotazioni

- `@Component`, nella quale viene definita la configurazione della classe.

View

La pagina HTML5 della schermata di gestione delle macchine comprende:

- un `button` per la creazione di una nuova macchina;
- una `mat-table` per la visualizzazione delle macchine. Per ogni macchina, ogni riga contiene:
 - il nome testuale;
 - un `button` per navigare al dettaglio della macchina;
 - un `button` per accendere o spegnere la ricezione della macchina, in base allo stato attuale;
 - un `button` per archiviare o ripristinare la macchina, in base allo stato attuale.
- elementi vari di design di Angular Material.

Interfacce implementate

- `OnInit`, richiesta per poter utilizzare il *life-cycle hook* `ngOnInit`.

Campi privati

- `router`: `Router` - permette la navigazione tra differenti URL;
- `deviceService`: `DeviceAbstractService` - permette di ottenere l'elenco delle macchine;
- `notificationService`: `NotificationService` - fornisce un feedback grafico agli utenti;
- `reloader`: `Observer<void>` - centralizza le operazioni di reload dopo una modifica.

Campi pubblici

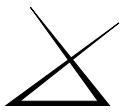
I campi pubblici della classe, a uso esclusivo del runtime di Angular, sono:

- `readonly displayedColumns: string[]` - rappresenta le colonne da visualizzare nella tabella;
- `devices: DeviceDataSource` - rappresenta l'elenco delle macchine da visualizzare nella tabella.

Metodi pubblici

I metodi pubblici della classe, a uso esclusivo del runtime di Angular, sono:

- `createDevice(): void`
Apre la pagina di inserimento di una nuova macchina (§4.1.8);
- `openDeviceDetail(
 – device: Device): void`
Apre la pagina di visualizzazione di dettaglio e modifica di una macchina (§4.1.9);
- `toggleActivationDevice()`



```
    - device: Device.  
  ): void  
  Permette di accendere o spegnere la ricezione della macchina passata come parametro;  
• toggleStatusDevice(  
  - device: Device.  
  ): void  
  Permette di modificare lo stato di archiviazione della macchina passata come parametro. Se la macchina non era archiviata, allora per procedere si dovrà interagire con la finestra di dialogo di conferma (§4.1.14.1).
```

4.1.7.2 DevicesModule

Questo modulo dichiara, esporta e definisce i percorsi URL di `DevicesComponent` e dei suoi figli, ognuno figlio di `/gestione-macchine`. Le route sono:

- `/`, `DevicesComponent`;
- `/nuova`, `NewDeviceComponent` (§4.1.8);
- `/:id`, dove `:id` è l'identificativo di una macchina: `DeviceDetailComponent` (§4.1.9).

4.1.7.3 DeviceDataSource

Questa classe rappresenta la sorgente per i dati da rappresentare nella `mat-table`.

Interfacce implementate

- `DataSource<Device>`, richiesta per poter essere utilizzata come sorgente dati di una `mat-table`.

Campi privati

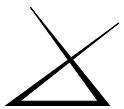
I campi privati sono:

- `dataStream: BehaviorSubject<Device[]>` - la sorgente asincrona di dati. In quanto `BehaviorSubject` dispone sempre di un valore. È inizializzato con un array vuoto.

Metodi pubblici

I metodi pubblici sono:

- `connect(): Observable<Device[]>`
Restituisce l'`Observable` che notificherà ogni cambio di valore;
- `disconnect(): void`
Disconnette la `mat-table` dalla sorgente;
- `setData(
 - data: Device[]): void`
Dispone un nuovo elenco di macchine.



4.1.7.4 DeviceAbstractService

Questa classe astratta modella le operazioni effettuate dalla schermata di gestione macchine.

Annotazioni

- `@Injectable`, per poter essere identificata all'interno del sistema di dependency injection di Angular.

Metodi pubblici

- `getDevices(): Observable<Device[]>`
Fornisce l'elenco di tutte le macchine censite nel sistema;
- `activateDevice(
 – device: Device: la macchina da attivare.
)`: `Observable<{}>`
Permette di attivare la macchina passata come parametro;
- `deactivateDevice(
 – device: Device: la macchina da disattivare.
)`: `Observable<{}>`
Permette di disattivare la macchina passata come parametro;
- `archiveDevice(
 – device: Device: la macchina da archiviare.
)`: `Observable<{}>`
Permette di archiviare la macchina passata come parametro;
- `restoreDevice(
 – device: Device: la macchina da ripristinare.
)`: `Observable<{}>`
Permette di ripristinare la macchina passata come parametro.

4.1.7.5 DeviceService

Questa classe gestisce l'interfacciamento delle richieste HTTP dal front-end verso il back-end.

Annotazioni

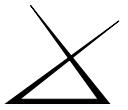
- `@Injectable`, che indica ad Angular di utilizzare questa classe nella dependency injection.

Interfacce implementate

- `DeviceAbstractService`

Campi privati

- `httpClient: HttpClient` - permette di effettuare richieste HTTP.



4.1.7.6 Device

Un oggetto che rispecchi questa interfaccia rappresenta una macchina del sistema. Viene utilizzata prevalentemente in fase di visualizzazione.

Campi pubblici

- `readonly id: number` - l'identificativo della macchina;
- `readonly name: string` - il nome della macchina;
- `readonly apiKey: string` - la chiave che identifica la macchina nell'API rilevazioni;
- `readonly archived: boolean` - se è true significa che la macchina è archiviata;
- `readonly deactivated: boolean` - se è true significa che la macchina è disattivata.

4.1.8 Schermata di creazione macchina

La pagina di creazione di una macchina è gestita dal modulo `src/app/admin/devices/devices.module.ts` (§4.1.7.2). Implementa l'interfaccia descritta in §3.4.1.1.7.

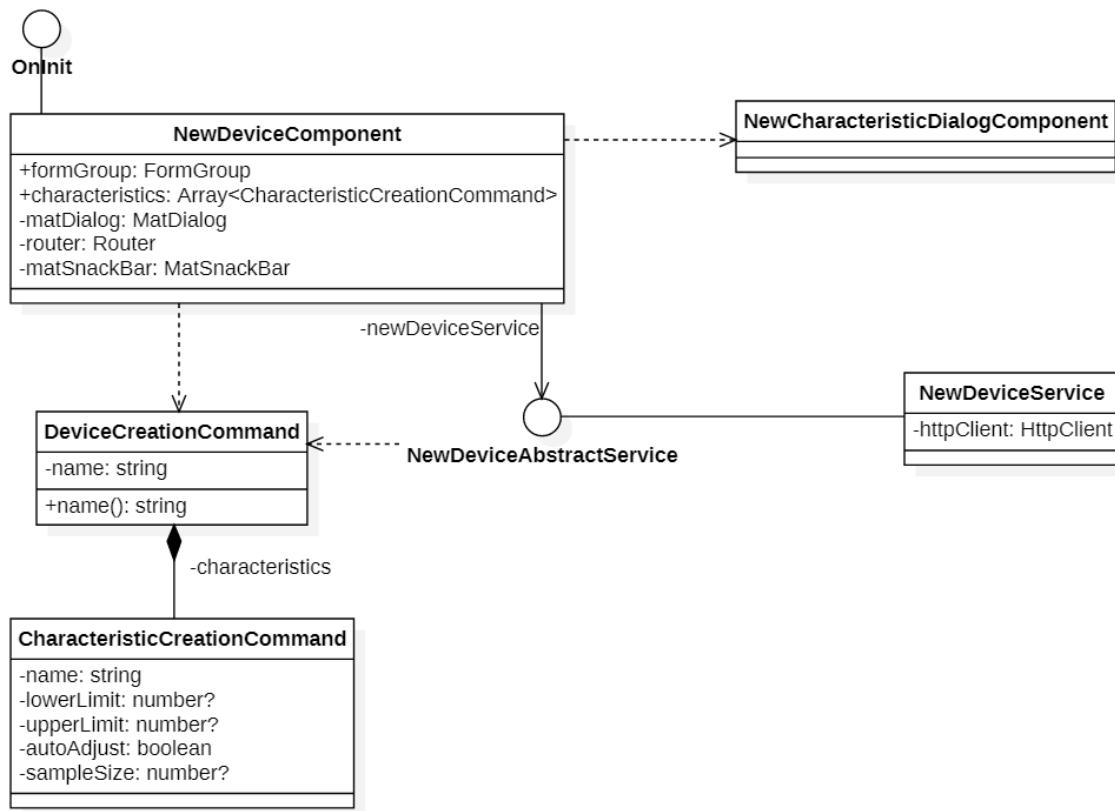
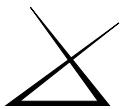


Figura 14: Diagramma delle classi della schermata di creazione macchina.



4.1.8.1 NewDeviceComponent

Annotazioni

- `@Component`, nella quale viene definita la configurazione della classe.

View

La pagina HTML5 della schermata di inserimento contiene:

- un `button` per annullare l'operazione e tornare all'elenco;
- un `button` per confermare i dati e procedere alla creazione della macchina;
- una `form` con i seguenti campi:
 - nome della macchina (include i messaggi d'errore sul campo).
- un `button` per aprire la schermata di inserimento di una nuova caratteristica;
- una `mat-list` per rappresentare le caratteristiche associate alla macchina.

Interfacce implementate

- `OnInit`, necessaria per agganciare il life-cycle hook `ngOnInit`.

Campi privati

- `matDialog: MatDialog` - permette di aprire finestre di dialogo;
- `newDeviceService: NewDeviceAbstractService` - permette di creare una nuova macchina;
- `router: Router` - permette di navigare tra le pagine dell'interfaccia;
- `notificationService: NotificationService` - permette un feedback grafico leggero all'utente.

Campi pubblici

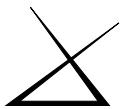
I campi pubblici della classe, a uso esclusivo del runtime di Angular, sono:

- `formGroup: FormGroup` - un oggetto che rappresenta una form reattiva di Angular. In questo oggetto vengono definiti i seguenti controlli:
 - `name`: un `FormControl` che gestisce la validazione del nome della macchina.
- `characteristics: Characteristic[]` - un array contenente le caratteristiche, che si vogliono associare alla macchina da creare;
- `readonly displayedColumns: string[]` - rappresenta le colonne da visualizzare nella tabella.

Metodi pubblici

I metodi pubblici della classe, a uso esclusivo del runtime di Angular, sono:

- `openNewCharacteristicDialog(): void`
Apre la finestra di dialogo di aggiunta di una caratteristica;
- `insertDevice(): void`
Tenta l'inserimento della macchina nel sistema;
- `removeCharacteristic(row: CharacteristicCreationCommand): void`
Annulla l'aggiunta di una caratteristica alla macchina in creazione.



4.1.8.2 NewDeviceAbstractService

Questa classe astratta modella le operazioni di inserimento di una nuova macchina tramite la relativa schermata.

Annotazioni

- `@Injectable`, che permette di utilizzare questo tipo all'interno della dependency injection.

Metodi pubblici

- `insertDevice()`
 - `device: DeviceCreationCommand`: le informazioni per l'inserimento.
- `): Observable<{id: number}>`
Tenta l'inserimento di una nuova macchina nel sistema.

4.1.8.3 NewDeviceService

Questa classe implementa le operazioni astratte di `NewDeviceAbstractService` effettuando richieste HTTP.

Annotazioni

- `@Injectable`, che permette di utilizzare questo tipo all'interno della dependency injection.
Inoltre, essa viene configurata in modo tale da costruire una sola istanza di questa classe.

Interfacce implementate

- `NewDeviceAbstractService`.

Campi privati

- `httpClient: HttpClient` - permette di effettuare richieste HTTP.

4.1.9 Schermata di modifica macchina

4.1.9.1 DeviceDetail

La pagina di modifica di una macchina è gestita dal modulo `src/app/admin/devices/devices.module.ts` (§4.1.7.2). Implementa l'interfaccia descritta in §3.4.1.1.8.

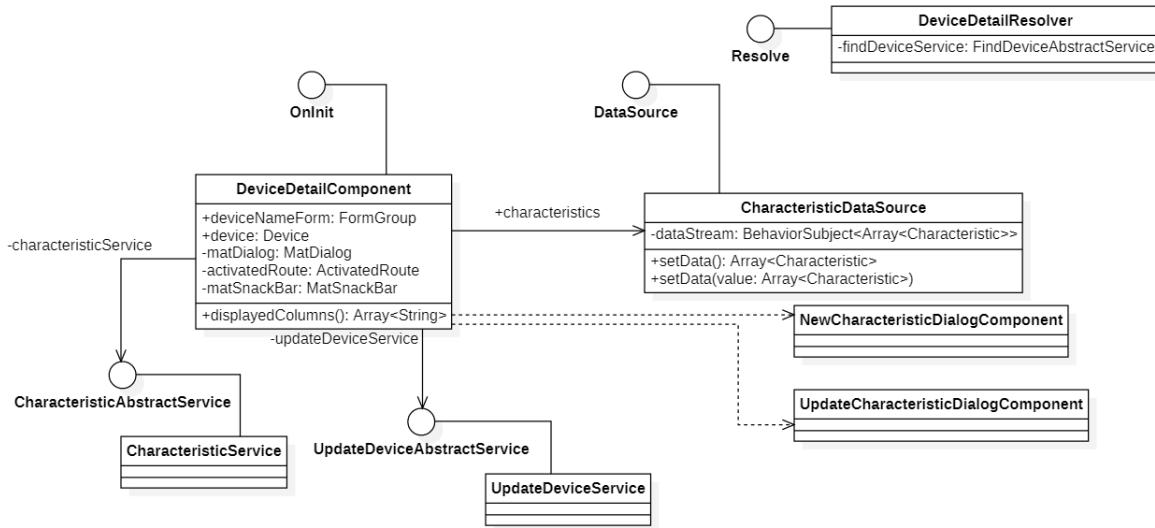
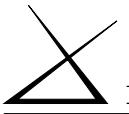


Figura 15: Diagramma delle classi della schermata di modifica macchina.

Annotazioni

- `@Component`, nel quale viene fornita la configurazione della classe.

View

La pagina HTML5 dell'interfaccia contiene:

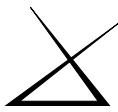
- il codice di accesso all'API rilevazioni della macchina;
- degli indicatori dello stato della macchina (archiviata, attiva, disattivata);
- un `button` che apre la finestra di dialogo di aggiunta di una caratteristica;
- una tabella di caratteristiche, per ognuna della quali:
 - il nome;
 - un `button` che permette di modificare lo stato di attivazione della caratteristica;
 - un `button` che apre la finestra di dialogo di modifica della caratteristica.

Interfacce implementate

- `OnInit`, necessaria per agganciare il life-cycle hook `ngOnInit`.

Campi privati

- `characteristicService: CharacteristicAbstractService` - permette di ottenere le caratteristiche data una macchina;
- `updateDeviceService: UpdateDeviceAbstractService` - permette di aggiornare una macchina;



- `matDialog: MatDialog` - permette di aprire finestre di dialogo;
- `activatedRoute: ActivatedRoute` - permette di accedere alle informazioni contenute nell'URL;
- `notificationService: NotificationService` - permette di fornire un feedback leggero all'utente.

Campi pubblici

I campi pubblici, a uso esclusivo del runtime di Angular, sono:

- `readonly displayedColumns: string[]` - espone i nomi delle colonne da mostrare nella `mat-table` che mostra le caratteristiche;
- `device: Device` - la macchina di cui si presentano i dati;
- `characteristics: CharacteristicsDatasource` - le caratteristiche della macchina;
- `deviceNameForm: FormGroup` - gestisce la validazione del nome della macchina.

Metodi pubblici

I metodi pubblici, a uso esclusivo del runtime di Angular, sono:

- `updateDeviceName(): void`
Permette di aggiornare il nome della macchina;
- `openNewCharacteristicDialog(): void`
Apre la finestra di dialogo che permette di inserire una nuova caratteristica;
- `openUpdateCharacteristicFormDialog(): void`
Apre la finestra di dialogo che permette di modificare una caratteristica esistente;
- `toggleCharacteristicStatus(): void`
Permette di modificare lo stato di archiviazione della caratteristica.

4.1.9.2 CharacteristicDataSource

Questa classe rappresenta la sorgente dati per la `mat-table` della pagina di dettaglio della macchina.

Interfacce implementate

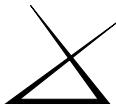
- `DataSource<Characteristic>`, richiesta per poter essere utilizzata in una `mat-table`.

Campi privati

- `dataStream: BehaviorSubject<Characteristic[]>` - modella la gestione asincrona dei dati da visualizzare. Permette sempre l'accesso all'ultimo dato segnalato.

Metodi pubblici

- `get data(): Characteristic[]`
L'ultimo elenco di caratteristiche emesso;
- `set data(`
 - `value: Characteristic[]`: i dati da visualizzare.`): void`
Aggiorna l'elenco di caratteristiche.



4.1.9.3 DeviceDetailResolver

Permette di effettuare un'interrogazione al back-end per ottenere le informazioni della macchina prima di caricare i dati.

Interfacce implementate

- `Resolve<Device>`, necessaria per poter utilizzare questa classe come resolver nella configurazione delle route.

Campi privati

- `findDeviceService: FindDeviceAbstractService` - permette di richiedere una macchina.

4.1.9.4 CharacteristicAbstractService

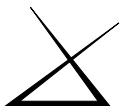
Questa classe astratta modella le operazioni dalla schermata di dettaglio e modifica di una macchina.

Annotazioni

- `@Injectable`, che permette di utilizzare questo tipo all'interno della dependency injection.

Metodi pubblici

- `abstract getCharacteristicsByDevice(`
 - `deviceId: number`: l'identificativo della macchina di cui richiedere le caratteristiche.`): Observable<Characteristic[]>`
Elenca le caratteristiche di una macchina dato il suo identificativo;
- `abstract addCharacteristic(`
 - `deviceId: number`: l'identificativo della macchina a cui aggiungere la caratteristica;
 - `command: CharacteristicCreationCommand`: le informazioni con cui inizializzare la nuova caratteristica.`): Observable<{ id: number }>`
Crea una nuova caratteristica per una data macchina. Restituisce un oggetto contenente l'identificativo progressivo della caratteristica creata;
- `abstract recoverCharacteristic(`
 - `deviceId: number`: l'identificativo della macchina della caratteristica da ripristinare;
 - `id: number`: l'identificativo della caratteristica.`): Observable<{}>`
Ripristina una caratteristica dati i suoi identificativi. Se la caratteristica non era archiviata, non succede nulla;
- `abstract archiveCharacteristic(`
 - `deviceId: number`: l'identificativo della macchina della caratteristica da archiviare;
 - `id: number`: l'identificativo della caratteristica.`): Observable<{}>`
Archivia una caratteristica dati i suoi identificativi. Se la caratteristica era già archiviata, non succede nulla.



4.1.9.5 CharacteristicService

Implementa le operazioni definite in `CharacteristicAbstractService` mediante richieste HTTP.

Annotazioni

- `@Injectable`, che permette di utilizzare questo tipo all'interno della dependency injection. Inoltre, essa viene configurata in modo tale da costruire una sola istanza di questa classe.

Interfacce implementate

- `CharacteristicAbstractService`.

Campi privati

- `httpClient: HttpClient` - permette di effettuare chiamate HTTP al back-end.

4.1.9.6 Characteristic

Campi pubblici

- `readonly id: number` - l'identificativo della caratteristica;
- `readonly name: string` - il nome della caratteristica;
- `readonly archived: boolean` - è true se e solo se la caratteristica è archiviata;
- `readonly autoAdjust: boolean` - è true se e solo se il calcolo automatico dei limiti è attivo;
- `readonly lowerLimit?: number` - (opzionale) il limite inferiore impostato dall'amministratore;
- `readonly upperLimit?: number` - (opzionale) il limite superiore impostato dall'amministratore;
- `readonly sampleSize?: number` - (opzionale) il numero di punti da considerare per calcolare i limiti, impostato dall'amministratore.

4.1.9.7 CharacteristicCreationCommand

- `readonly name: string` - il nome della caratteristica da creare;
- `readonly autoAdjust: boolean` - è true se e solo se il calcolo automatico dei limiti è attivo;
- `readonly lowerLimit?: number` - (opzionale) il limite inferiore impostato dall'amministratore;
- `readonly upperLimit?: number` - (opzionale) il limite superiore impostato dall'amministratore;
- `readonly sampleSize?: number` - (opzionale) il numero di punti da considerare per calcolare i limiti, impostato dall'amministratore.

4.1.9.8 FindDeviceAbstractService

Questa classe astratta modella la richiesta di informazioni dettagliate su una macchina in particolare, per la schermata di dettaglio macchina.

Annotazioni

- `@Injectable`, che permette di utilizzare questo tipo all'interno della dependency injection.

Metodi pubblici

- `findDeviceById(`
 - `id: number`: l'identificativo della macchina.
- `): Observable<Device>`
Richiede le informazioni di una macchina. Restituisce un osservabile che notifica quando le informazioni sono pronte.

4.1.9.9 FindDeviceService

Questa classe implementa le operazioni astratte di `FindDeviceAbstractService` effettuando richieste HTTP.

Annotazioni

- `@Injectable`, che permette di utilizzare questo tipo all'interno della dependency injection.
Inoltre, essa viene configurata in modo tale da costruire una sola istanza di questa classe.

Interfacce implementate

- `FindDeviceAbstractService`.

Campi privati

- `httpClient: HttpClient` - permette di effettuare richieste HTTP.

4.1.9.10 UpdateDeviceAbstractService

Questa classe astratta modella l'aggiornamento delle informazioni della macchina possibili dalla schermata di dettaglio macchine.

Annotazioni

- `@Injectable`, che permette di utilizzare questo tipo all'interno della dependency injection.

Metodi pubblici

- `updateDeviceName(`
 - `id: number`: l'identificativo della macchina;
 - `newName: string`: il nuovo nome della macchina.
- `): Observable<{}>`
Cambia il nome della macchina.

4.1.9.11 UpdateDeviceService

Questa classe implementa le operazioni astratte di `UpdateDeviceAbstractService` effettuando richieste HTTP.

Annotazioni

- `@Injectable`, che permette di utilizzare questo tipo all'interno della dependency injection.
Inoltre, essa viene configurata in modo tale da costruire una sola istanza di questa classe.

Interfacce implementate

- UpdateDeviceAbstractService.

Campi privati

- httpClient: HttpClient - permette di effettuare richieste HTTP.

4.1.10 Form per la creazione di una caratteristica

4.1.10.1 NewCharacteristicDialogComponent

Questa classe rappresenta una finestra di dialogo gestita dal modulo `src/app/admin/devices/devices.module.ts` (§4.1.7.2). Implementa l'interfaccia descritta in §3.4.1.1.9.

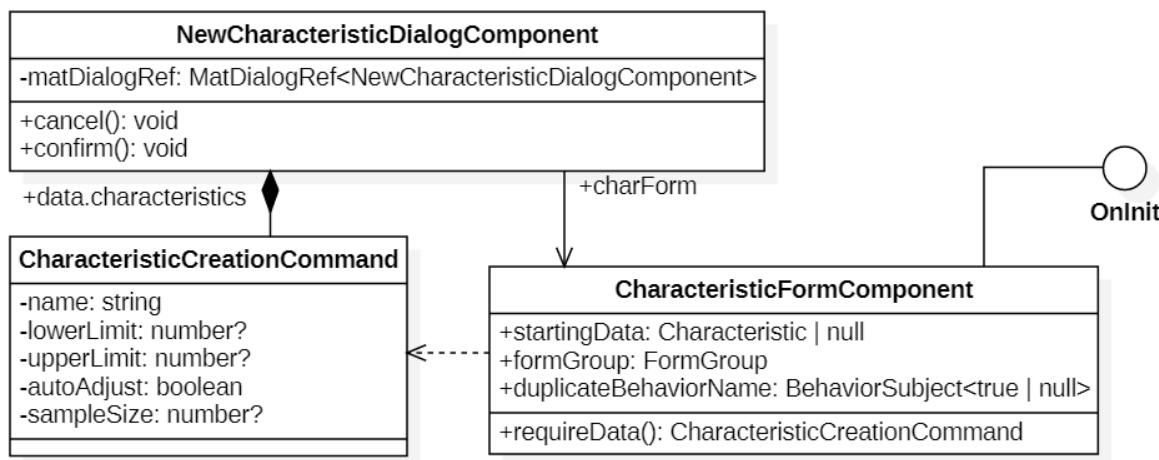


Figura 16: Diagramma delle classi della finestra di dialogo di creazione caratteristica.

Annotazioni

- `@Component`, nella quale viene definita la configurazione del component.

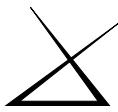
View

La pagina HTML5 correlata contiene:

- una `app-characteristic-form`, ovvero un componente che centralizza la logica di modifica e inserimento di una caratteristica;
- un `button` di conferma.

Campi privati

- `matDialogRef: MatDialogRef<NewCharacteristicDialogComponent>` - il riferimento al `MatDialog`, che permette, tra le varie operazioni, di chiuderlo.



Campi pubblici

I campi pubblici, a uso esclusivo del runtime di Angular, sono:

- `data: { characteristics: CharacteristicCreationCommand[] }` - contiene i dati utili alla validazione della nuova caratteristica. Viene fornito al momento della creazione della finestra di dialogo;
- `@ViewChild(`
 - `'charForm'` - il nome dell'ancora alla form delle caratteristiche.
-) `charForm: CharacteristicFormComponent` - un puntatore al `CharacteristicFormComponent` contenuto nell'HTML.

Metodi pubblici

I metodi pubblici, a uso esclusivo del runtime di Angular, sono:

- `cancel(): void`
Chiude la finestra di dialogo annullando l'operazione;
- `confirm(): void`
Chiude la finestra di dialogo confermando i dati.

4.1.10.2 CharacteristicFormComponent

Annotazioni

- `@Component`, che contiene la configurazione del componente.

View

La pagina HTML5 che costituisce la vista contiene una `form` che modella i seguenti campi di una caratteristica:

- nome;
- calcolo automatico dei limiti, come un `mat-checkbox`;
- se il calcolo automatico è disabilitato, limite inferiore e limite superiore.

Interfacce implementate

- `OnInit`, necessaria per agganciare il life-cycle hook `ngOnInit`. In questa classe, viene utilizzato per configurare la validazione del form.

Campi pubblici

- `@Input() startingData: Characteristic` - se `null`, significa che il form deve creare una caratteristica da zero. Altrimenti, utilizza quei dati per inizializzare il form. Questo valore può essere inizializzato dall'HTML;
- `formGroup: FormGroup` - rappresenta il gruppo di campi del form;
- `duplicateNameBehavior: BehaviorSubject<true | null>` - permette all'esterno di notificare un errore al form, per mostrare all'utente che il nome è duplicato.

Metodi pubblici

- `requireData(): CharacteristicCreationCommand` - permette di ottenere l'oggetto che rappresenta i dati con cui inserire la caratteristica.

4.1.11 Form per la modifica di una caratteristica

4.1.11.1 UpdateCharacteristicDialogComponent

Questa classe rappresenta una finestra di dialogo gestita dal modulo `src/app/admin/devices/devices.module.ts` (§4.1.7.2). Implementa l'interfaccia descritta in §3.4.1.1.10.

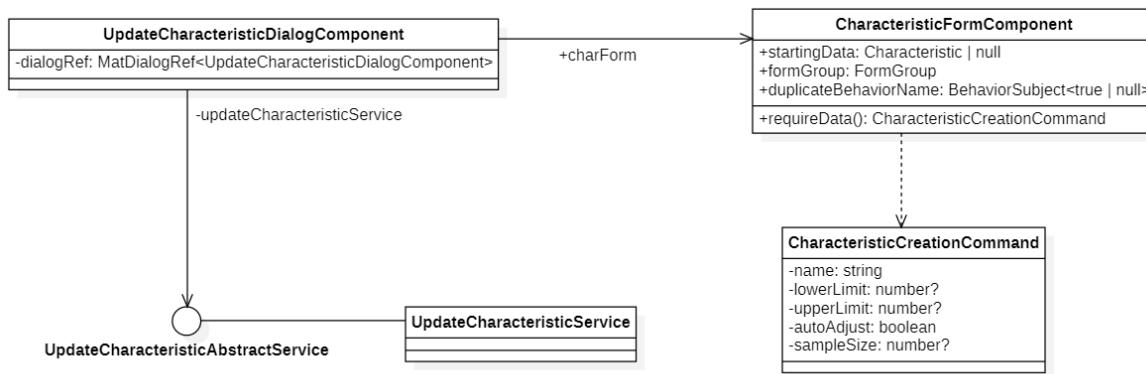


Figura 17: Diagramma delle classi della finestra di dialogo di modifica caratteristica.

Annotazioni

- `@Component`, che contiene la configurazione del componente.

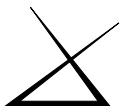
View

La pagina HTML5 annessa a questo component contiene:

- un `app-characteristic-form`, che rappresenta una form per gestire la modifica di una caratteristica;
- un `button` per annullare l'operazione;
- un `button` per confermare l'operazione.

Campi privati

- `updateCharacteristicService: UpdateCharacteristicAbstractService` - permette di effettuare l'aggiornamento dei dati di una caratteristica;
- `dialogRef: MatDialogRef<UpdateCharacteristicDialogComponent>` - il riferimento alla finestra di dialogo che permette, tra le altre cose, di poterla chiudere.



Campi pubblici

I campi pubblici, a uso esclusivo del runtime di Angular, sono:

- `@ViewChild(`
 - ‘charForm’.
-) `charForm: CharacteristicFormComponent` - un riferimento alla `app-characteristic-form` in interfaccia;
- `public data: {deviceId: number, characteristic: Characteristic}` - i dati necessari a identificare la caratteristica. Viene fornito al momento della creazione della finestra di dialogo.

Metodi pubblici

- `cancel(): void`
Chiude la finestra di dialogo annullando l'operazione;
- `confirm(): void`
Tenta l'inserimento dei dati al back-end e, se l'esito è positivo, chiude la finestra di dialogo.

4.1.11.2 UpdateCharacteristicAbstractService

Questa classe astratta modella le operazioni effettuabili dalla schermata di modifica di una caratteristica.

Annotazioni

- `@Injectable`, che permette di utilizzare questo tipo all'interno della dependency injection.

Metodi pubblici

- `abstract updateCharacteristic(`
 - `command: CharacteristicUpdateCommand.`
-) `: Observable<{}>`
Aggiorna le informazioni di una caratteristica.

4.1.11.3 UpdateCharacteristicService

Implementa le operazioni astratte descritte in `UpdateCharacteristicAbstractService` eseguendo delle richieste HTTP al back-end.

Annotazioni

- `@Injectable`, che permette di utilizzare questo tipo all'interno della dependency injection.
Inoltre, essa viene configurata in modo tale da costruire una sola istanza di questa classe.

Interfacce implementate

- `UpdateCharacteristicAbstractService.`

Campi privati

- `httpClient: HttpClient` - permette di effettuare richieste HTTP.

4.1.11.4 CharacteristicUpdateCommand

- `readonly deviceId: number` - l'identificativo della macchina;
- `readonly id: number` - l'identificativo della caratteristica;
- `readonly name: string` - il nome della caratteristica da creare;
- `readonly autoAdjust: boolean` - è true se e solo se il calcolo automatico dei limiti è attivo;
- `readonly lowerLimit?: number` - (opzionale) il limite inferiore impostato dall'amministratore;
- `readonly upperLimit?: number` - (opzionale) il limite superiore impostato dall'amministratore;
- `readonly sampleSize?: number` - (opzionale) il numero di punti da considerare per calcolare i limiti, impostato dall'amministratore.

4.1.12 Schermata di gestione utenti

4.1.12.1 AccountsComponent

Questa classe rappresenta una pagina gestita dal modulo `src/app/admin/accounts/accounts.module.ts` (§4.1.12.2). Implementa l'interfaccia descritta in §RIF21.

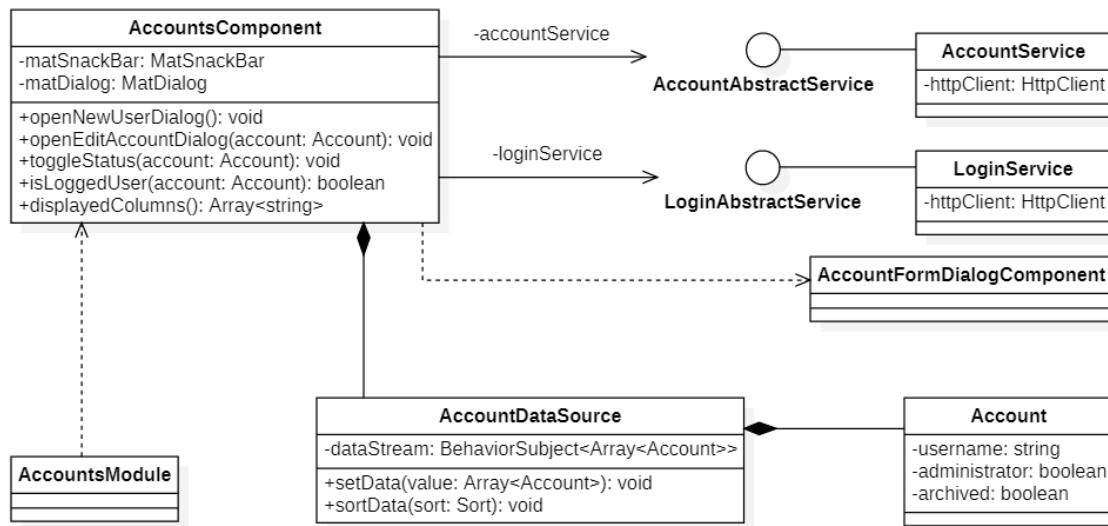


Figura 18: Diagramma delle classi della finestra di lista utenti.

Annotazioni

- `@Component`, nella quale viene definita la configurazione della classe.

View

La pagina HTML5 di questo componente contiene:

- un button per registrare un nuovo utente;

- una **mat-table** che mostra gli utenti registrati. Ogni riga contiene:
 - l'username dell'utente in testo semplice;
 - una **mat-icon** che indica se l'utente è amministratore;
 - un **button** per disabilitare l'utente, se era abilitato;
 - un **button** per abilitare l'utente, se era disabilitato.

Interfacce implementate

- **OnInit**, necessaria per agganciare il life-cycle hook **ngOnInit**.

Campi privati

- **accountService**: `AccountAbstractService` - permette di effettuare ricerche tra gli utenti censiti dal sistema;
- **matDialog**: `MatDialog` - permette di aprire finestre di dialogo;
- **notificationService**: `NotificationService` - permette un rapido feedback visivo all'utente;
- **loginService**: `LoginAbstractService` - contiene le informazioni della sessione.

Campi pubblici

- **accounts**: `AccountsDataSource` - la classe che modella i dati da visualizzare nella **mat-table**.

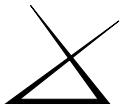
Metodi pubblici

- **openNewAccountDialog(): void**
Apre la finestra di dialogo per creare un nuovo utente;
- **openEditAccountDialog(): void**
Apre la finestra di modifica di un utente;
- **toggleStatus()**
 - **account**: `Account`: l'account da modificare.
- **loggedUser()**
 - **account**: `Account`: l'account in esame.

4.1.12.2 AccountsModule

Questo modulo gestisce l'URL di `AccountsComponent`:

- `/gestione-utenti`: `AccountsComponent`.



4.1.12.3 AccountsDataSource

Interfacce implementate

- `DataSource<Account>`, necessaria per poter utilizzare la classe come sorgente dati per la `mat-table`.

Campi privati

- `dataStream: BehaviorSubject<Account[]>` - notifica l'aggiornamento dei dati.

Metodi pubblici

- `setData(`
 - `data: Account[]`: gli account da visualizzare.`): void`
Aggiorna i dati da mostrare;
- `sortData(`
 - `sort: Sort`: le indicazioni di ordinamento.`): void`
Ordina i dati da mostrare.

4.1.12.4 AccountAbstractService

Questa classe astratta modella le operazioni sugli utenti registrati possibili dalla schermata di gestione utenti.

Annotazioni

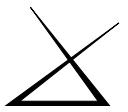
- `@Injectable`, che permette di utilizzare questo tipo all'interno della dependency injection.

Metodi pubblici

- `getAccounts(): Observable<Account[]>`
Ottiene l'elenco degli utenti registrati e le loro informazioni;
- `archiveAccount(`
 - `account: Account`: l'utente da archiviare.`): Observable<{}>`
Archivia l'utente passato come parametro. Se l'utente era già archiviato, non succede nulla;
- `recoverAccount(`
 - `account: Account`: l'utente da ripristinare.`): Observable<{}>`
Ripristina l'utente passato come parametro. Se l'utente non era archiviato, non succede nulla.

4.1.12.5 AccountService

Questa classe implementa le operazioni astratte di `AccountAbstractService` effettuando richieste HTTP.



Annotazioni

- `@Injectable`, che permette di utilizzare questo tipo all'interno della dependency injection. Inoltre, essa viene configurata in modo tale da costruire una sola istanza di questa classe.

Interfacce implementate

- `AccountAbstractService`.

Campi privati

- `httpClient: HttpClient` - permette di effettuare richieste HTTP.

4.1.12.6 Account

Campi pubblici

- `readonly username: string` - il nome dell'utente;
- `readonly administrator: boolean` - è true se e solo se l'utente è amministratore;
- `readonly archived: boolean` - è true se e solo se l'utente è archiviato.

4.1.13 Form di creazione e modifica utente

Questa schermata implementa le interfacce descritte in §3.4.1.1.12 e §3.4.1.1.13. L'interfaccia mostrata e l'azione richiesta sono configurabili.

AccountFormDialogComponent

Un componente che incapsula la logica di salvataggio di un account.

Annotazioni

- `@Component`, dove viene fornita la configurazione come componente.

View

La pagina HTML5 che costituisce la View del componente comprende:

- una `form`, contenente i seguenti campi:
 - nome utente;
 - se l'utente sia amministratore o no;
 - in modalità di modifica:
 - * un controllo che indica se cambiare o meno la password;
 - * se il checkbox sopracitato è attivo, la password.
 - in modalità di creazione, la password.
- un `button` per annullare l'operazione;
- un `button` per confermare l'operazione.

Interfacce implementate

- `OnInit`, necessaria per agganciare il life-cycle hook `ngOnInit`, usato per configurare i validatori del form.

Campi privati

- `matDialogRef: MatDialogRef<AccountFormDialogComponent>` - riferimento alla finestra di dialogo;
- `saveAccountService: SaveAccountAbstractService` - permette di salvare le informazioni dell'utente;
- `notificationService: NotificationService` - permette un feedback visivo rapido;
- `passwordValidators: ValidatorFn[]` - i validatori per il campo della password.

Campi pubblici

- `formGroup: FormGroup` - l'oggetto che modella i campi della form;
- `data: account: Account | null`

Metodi pubblici

- `editMode(): boolean`
Restituisce `true` se l'istanza è configurata in inserimento, altrimenti `false` se configurata in modifica;
- `cancel(): void`
Chiude la finestra di dialogo, annullando l'operazione;
- `confirm(): void`
Tenta l'inserimento dei dati e, se ha successo, chiude la finestra di dialogo;
- `toggleChangePassword()`
 - `enabled: boolean` - se la password stia venendo cambiata o se bisogni mantenere la password attuale.
- `): void`
Attiva il cambio password, altrimenti viceversa.

4.1.13.1 SaveAccountAbstractService

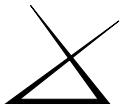
Questa classe astratta modella le operazioni sugli utenti registrati dalla schermata di creazione e salvataggio utente.

Annotazioni

- `@Injectable`, che permette di utilizzare questo tipo all'interno della dependency injection.

Metodi pubblici

- `insertAccount()`
 - `command: AccountSaveCommand`: le informazioni con cui inizializzare l'utente da creare.
- `): Observable<{ username: string }>`
Crea un nuovo utente utilizzando le informazioni contenute nel parametro. Restituisce un osservabile che notifica fornendo l'username dell'utente appena creato;
- `updateAccount()`



- `command: AccountSaveCommand`: le informazioni aggiornate di un utente.
-) : `Observable<{}>`
Aggiorna le informazioni di un utente.

4.1.13.2 SaveAccountService

Questa classe implementa le operazioni astratte definite in `SaveAccountAbstractService` effettuando delle richieste HTTP.

Annotazioni

- `@Injectable`, che permette di utilizzare questo tipo all'interno della dependency injection.
Inoltre, essa viene configurata in modo tale da costruire una sola istanza di questa classe.

Interfacce implementate

- `SaveAccountAbstractService`.

Campi privati

- `httpClient: HttpClient` - permette di effettuare richieste HTTP.

4.1.13.3 AccountSaveCommand

- `readonly username: string` - il nome dell'utente;
- `readonly password?: string` - (opzionale) la password dell'utente. Se la password non viene fornita in modifica, essa non viene cambiata in banca dati. Se la password non viene fornita in creazione, viene visualizzato un errore a schermo;
- `readonly administrator: boolean` - è true se e solo se l'utente è un amministratore del sistema *Produlytics*.

4.1.14 Altre componenti

Qui vengono riportate altri componenti utilizzati in tutto il front-end.

4.1.14.1 ConfirmDialogComponent

Una finestra di dialogo generica per chiedere la conferma di un'operazione.

Annotazioni

- `@Component`, che configura la classe come un component di Angular.

View

La pagina HTML5 di questo component contiene:

- un messaggio testuale che descrive l'azione;
- un `button` per confermare l'azione;
- un `button` per annullare l'azione.

Campi pubblici

- `data: {message: string}` - contiene la domanda di conferma da visualizzare a schermo.

Campi privati

- `matDialogRef: MatDialogRef<ErrorDialogComponent>` - permette di chiudere la finestra di dialogo.

Metodi pubblici

- `close(): void`
Chiude la finestra di dialogo annullando l'operazione;
- `confirm(): void`
Chiude la finestra di dialogo confermando l'operazione.

4.1.14.2 ErrorDialogComponent

Un component per mostrare errori imprevisti.

Annotazioni

- `@Component`, che configura la classe come un component di Angular.

View

La pagina HTML5 di questo component contiene:

- un messaggio d'errore;
- un button per chiudere la finestra di dialogo.

Campi pubblici

- `data: {message: string}` - contiene il messaggio da visualizzare a schermo.

Campi privati

- `matDialogRef: MatDialogRef<ErrorDialogComponent>` - permette di chiudere la finestra di dialogo.

Metodi pubblici

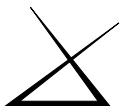
- `close(): void`
Chiude la finestra di dialogo.

4.1.15 Classi di utilità

Questa sottosezione raccoglie le classi di utilità generiche utilizzate in tutto il progetto.

4.1.15.1 NotificationService

Questa classe raccoglie la configurazione di diversi modi di notificare azioni all'utente.



Annotazioni

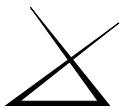
- `@Injectable`, per poter essere identificato all'interno del sistema di dependency injection di Angular.

Campi privati

- `matSnackBar: MatSnackBar` - permette la visualizzazione di messaggi di feedback;
- `dialog: MatDialog` - permette l'apertura di finestre di dialogo.

Metodi pubblici

- `notify(): void`
Fa apparire in basso al centro un messaggio di feedback;
- `unexpectedError(message: string): void`
Fa apparire una finestra di dialogo con un errore;
- `requireConfirm(message: string): Observable<boolean>`
Fa apparire una finestra di dialogo con una domanda e due buttoni: uno per confermare e l'altro per annullare. La funzione restituisce un `Observable` con valore `true` o `false`, in base alla scelta dell'utente.



4.2 Back-end UI

Come la componente API rilevazioni, anche la componente back-end UI è sviluppata in Java utilizzando il framework Spring Boot versione 2.6.2. La progettazione segue il pattern dell'architettura esagonale. I file sorgenti sono raggruppati nella cartella `src` e organizzati secondo la seguente struttura:

- `main/java`: contiene i sorgenti del software che sarà eseguito;
- `main/resources`: contiene la configurazione del software che andrà eseguito;
- `test/java`: contiene i sorgenti dei test;
- `test/resources`: contiene la configurazione del software da usare durante i test.

In particolare la cartella `main/java` contiene il package `it.deltax.prodlytics.backend`, il quale contiene a sua volta i package:

- `repositories`: contiene le interfacce relative alla comunicazione con il database;
- `exceptions`: contiene le classi relative alla gestione delle eccezioni e degli errori;
- `accounts`: contiene le classi, interfacce e record relativi agli endpoint sotto `/accounts`, oltre che alle classi per la configurazione;
- `admins`: contiene le classi, interfacce e record relativi agli endpoint sotto `/admins`, oltre che alle classi per la configurazione. Esso contiene a sua volta i package:
 - `accounts`: contiene le classi, interfacce e record relativi agli endpoint sotto `/admins` e riguardanti gli utenti;
 - `devices`: contiene le classi, interfacce e record relativi agli endpoint sotto `/admins` e riguardanti le macchine o le caratteristiche.
- `detections`: contiene le classi, interfacce e record relativi agli endpoint sotto `/devices` e riguardanti le rilevazioni, oltre che alle classi per la configurazione;
- `devices`: contiene le classi, interfacce e record relativi agli endpoint sotto `/devices` e riguardanti le macchine o le caratteristiche, oltre che alle classi per la configurazione;
- `security`: contiene le classi, interfacce e record relativi alla configurazione di Spring Security, quindi all'autenticazione e al controllo dei permessi.

I package `accounts`, `admins.accounts`, `admins.devices`, `detections` e `devices` al loro interno sono organizzati secondo la struttura:

- `web`: contiene le classi relative ai controller, ovvero quelle con il compito di ricevere le richieste fatte all'endpoint;
- `adapters`: contiene le classi il cui compito è adattare i repository alle interfacce di dominio;
- `business`: contiene le classi e interfacce di business, indipendenti dal framework usato, a loro volta divise in:
 - `ports/in`: contiene le interfacce dei casi d'uso forniti dalla logica di business;
 - `ports/out`: contiene le interfacce delle porte richieste dalla logica di business;
 - `services`: contiene l'implementazione dei servizi di business;
 - `domain`: contiene tutti i record utilizzati all'interno dei servizi di business.

La cartella `test/java` contiene il package `it.deltax.produlytics.backend`, il quale contiene i package `unit` e `integration`, che dividono i test di unità da quelli di integrazione.

Per ogni classe in questa sezione sono impliciti:

- l'esistenza di un costruttore che accetta un parametro per campo e lo assegna al relativo campo, a meno che non siano definiti altri costruttori;
- l'esistenza di un getter per ogni campo di un record, con lo stesso nome del relativo campo.

4.2.1 Controller

Di seguito vengono elencate tutte le classi con il ruolo di controller di Spring Boot:

4.2.1.1 AccountsController

Questa classe è un controller di Spring Boot. Il suo ruolo è essere il punto d'entrata delle richieste HTTP effettuate dagli utenti e inoltrarle ai service della parte di business.

Annotazioni

- `@RestController`;
- `@RequestMapping("/accounts")`.

Campi privati

- `UpdateAccountPasswordUseCase updateAccountPasswordUseCase`: l'istanza di un servizio a cui inoltrare le richieste per l'aggiornamento della password.

Metodi pubblici

- `@PutMapping("/{username}/password") ResponseEntity<String> updateAccountPassword(@PathVariable("username") String username, @RequestBody JsonNode body)`: il corpo della richiesta HTTP, in formato JSON.
Questo metodo implementa l'endpoint descritto in §3.4.2.1.4.

4.2.1.2 AdminsAccountsController

Questa classe è un controller di Spring Boot. Il suo ruolo è essere il punto d'entrata delle richieste HTTP effettuate dagli amministratori e relative alla gestione degli utenti e inoltrarle ai service della parte di business.

Annotazioni

- `@RestController`;
- `@RequestMapping("/admin")`.

Campi privati

- `InsertAccountUseCase insertAccountUseCase`: l'istanza di un servizio a cui inoltrare le richieste per l'inserimento di un nuovo utente;
- `GetTinyAccountsUseCase getTinyAccountsUseCase`: l'istanza di un servizio a cui inoltrare le richieste per l'ottenimento della lista degli utenti, ognuno con le informazioni essenziali;
- `UpdateAccountByAdminUseCase updateAccountByAdminUseCase`: l'istanza di un servizio a cui inoltrare le richieste per l'aggiornamento di un utente;
- `UpdateAccountArchiveStatusUseCase updateAccountArchiveStatusUseCase`: l'istanza di un servizio a cui inoltrare le richieste per l'aggiornamento dello stato di archiviazione di un utente.

Metodi pubblici

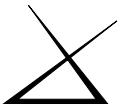
- `@PostMapping("/accounts") ResponseEntity<Map<String, String>> insertAccount(@RequestBody AccountToInsert account)`
Questo metodo implementa l'endpoint descritto in §3.4.2.1.21;
`) throws BusinessException`
Questo metodo implementa l'endpoint descritto in §3.4.2.1.21;
- `@GetMapping("/accounts") ResponseEntity<List<TinyAccount>> getAccounts()`
throws `BusinessException`
Questo metodo implementa l'endpoint descritto in §3.4.2.1.19;
- `@PutMapping("/{username}") ResponseEntity<String> updateAccount(@PathVariable("username") String username, @RequestBody AccountDataToUpdate body)`
Questo metodo implementa l'endpoint descritto in §3.4.2.1.22;
`) throws BusinessException`
Questo metodo implementa l'endpoint descritto in §3.4.2.1.22;
- `@PutMapping("accounts/{username}/archived") ResponseEntity<String> updateAccountArchiveStatus(@PathVariable("username") String username, @RequestBody JsonNode body)`
Questo metodo implementa l'endpoint descritto in §3.4.2.1.20.

4.2.1.3 AdminsDevicesController

Questa classe è un controller di Spring Boot. Il suo ruolo è essere il punto d'entrata delle richieste HTTP effettuate dagli amministratori e relative alla gestione delle macchine e inoltrarle ai service della parte di business.

Annotazioni

- `@RestController;`
- `@RequestMapping("/admin").`



Campi privati

- `InsertDeviceUseCase insertDeviceUseCase`: l'istanza di un servizio a cui inoltrare le richieste per l'inserimento di una nuova macchina;
- `GetDevicesUseCase getDevicesUseCase`: l'istanza di un servizio a cui inoltrare le richieste per l'ottenimento della lista delle macchine;
- `GetDeviceDetailsUseCase getDeviceDetailsUseCase`: l'istanza di un servizio a cui inoltrare le richieste per l'ottenimento dei dettagli di una macchina;
- `UpdateDeviceNameUseCase updateDeviceNameUseCase`: l'istanza di un servizio a cui inoltrare le richieste per l'aggiornamento del nome di una macchina;
- `UpdateDeviceArchiveStatusUseCase updateDeviceArchiveStatusUseCase`: l'istanza di un servizio a cui inoltrare le richieste per l'aggiornamento dello stato di archiviazione di una macchina;
- `@PostMapping("/devices") ResponseEntity<Map<String, String>> insertDevice(@RequestBody DeviceToInsert device)`: un record contenente i dati della macchina da inserire.
Questo metodo implementa l'endpoint descritto in §3.4.2.1.12;
- `@GetMapping("/devices") ResponseEntity<Iterable<Device>> getDevices()`
throws `BusinessException`
Questo metodo implementa l'endpoint descritto in §3.4.2.1.9;
- `@GetMapping("/devices/{id}") ResponseEntity<Optional<DetailedDevice>> getDeviceDetails(@PathVariable("id") int id)`: l'identificativo della macchina di cui devono essere restituiti i dettagli.
Questo metodo implementa l'endpoint descritto in §3.4.2.1.13;
- `@PutMapping("/devices/{id}/name") ResponseEntity<String> updateDeviceName(@PathVariable("id") int id, @RequestBody JsonNode body)`: il identificativo della macchina da aggiornare; il body della richiesta, in formato JSON.
Questo metodo implementa l'endpoint descritto in §3.4.2.1.14;
- `@PutMapping("/devices/{id}/archived") ResponseEntity<String> updateDeviceArchiveStatus(@PathVariable("id") int id, @RequestBody JsonNode body)`: il identificativo della macchina di cui aggiornare lo stato di archiviazione; il corpo della richiesta HTTP, in formato JSON.

```
) throws BusinessException  
Questo metodo implementa l'endpoint descritto in §3.4.2.1.10;  
  
• @PutMapping("/devices/{id}/deactivated") ResponseEntity<String>  
    updateDeviceDeactivateStatus(  
  
        @PathVariable("id") int id: l'identificativo della macchina di cui aggiornare lo stato di  
        attivazione;  
        @RequestBody JsonNode body: il body della richiesta, in formato JSON.  
  
) throws BusinessException  
Questo metodo implementa l'endpoint descritto in §3.4.2.1.11.
```

4.2.1.4 DevicesController

Questa classe è un controller di Spring Boot. Il suo ruolo è essere il punto d'entrata delle richieste HTTP effettuate, relative alle macchine non archiviate e inoltrarle ai service della parte di business.

Annotazioni

- @RestController;
- @RequestMapping("/devices").

Campi privati

- GetUnarchivedDevicesUseCase getUnarchivedDevicesUseCase: l'istanza di un servizio a cui inoltrare le richieste per l'ottenimento dell'elenco delle macchine non archiviate.

Metodi pubblici

- @GetMapping("") ResponseEntity<Iterable<TinyDevice>> getUnarchivedDevices()
) throws BusinessException
 Questo metodo implementa l'endpoint descritto in §3.4.2.1.5.

4.2.1.5 AdminsCharacteristicsController

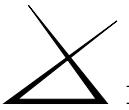
Questa classe è un controller di Spring Boot. Il suo ruolo è essere il punto d'entrata delle richieste HTTP effettuate dagli amministratori e relative alle caratteristiche di una macchina e inoltrarle ai service della parte di business.

Annotazioni

- @RestController;
- @RequestMapping("/admin/devices/{deviceId}/characteristics").

Campi privati

- InsertCharacteristicUseCase insertCharacteristicUseCase: l'istanza di un servizio a cui inoltrare le richieste per l'inserimento di una nuova caratteristica;
- GetCharacteristicsUseCase getCharacteristicsUseCase: l'istanza di un servizio a cui inoltrare le richieste per l'ottenimento delle caratteristiche di una macchina;



- `UpdateCharacteristicArchiveStatusUseCase updateCharacteristicArchiveStatusUseCase`: l'istanza di un servizio a cui inoltrare le richieste per la modifica dello stato di archiviazione della caratteristica di una macchina;
- `UpdateCharacteristicUseCase updateCharacteristicUseCase`: l'istanza di un servizio a cui inoltrare le richieste per la modifica della caratteristica di una macchina.

Metodi pubblici

- ```
@PostMapping("") ResponseEntity<Map<String, Integer>> insertCharacteristic(@PathVariable("deviceId") int deviceId: l'identificativo della macchina; @RequestBody NewCharacteristic characteristic: le informazioni della nuova caratteristica.) throws BusinessException
```

Questo metodo implementa l'endpoint descritto in §3.4.2.1.17;
- ```
@GetMapping("") List<Characteristic> getCharacteristics(@PathVariable("deviceId") int deviceId: l'identificativo della macchina.) throws BusinessException
```

Questo metodo implementa l'endpoint descritto in §3.4.2.1.15;
- ```
@PutMapping("/{characteristicId}/archived") ResponseEntity<String> updateCharacteristicArchiveStatus(@PathVariable("deviceId") int deviceId: l'identificativo della macchina; @PathVariable("characteristicId") int characteristicId: l'identificativo, relativo alla macchina, della caratteristica da modificare; @RequestBody JsonNode body: il corpo della richiesta HTTP, in formato JSON.) throws BusinessException
```

Questo metodo implementa l'endpoint descritto in §3.4.2.1.16;
- ```
@PutMapping("/{characteristicId}") ResponseEntity<String> updateCharacteristic(@PathVariable("deviceId") int deviceId: l'identificativo della macchina; @PathVariable("characteristicId") int characteristicId: l'identificativo della caratteristica, relativo alla macchina; @RequestBody JsonNode body: il corpo della richiesta HTTP, in formato JSON.) throws BusinessException
```

Questo metodo implementa l'endpoint descritto in §3.4.1.1.10.

4.2.1.6 CharacteristicsController

Questa classe è un controller di Spring Boot. Il suo ruolo è essere il punto d'entrata delle richieste HTTP effettuate, relative alle caratteristiche non archiviate di una macchina non archiviata e inoltrarle ai service della parte di business.

Annotazioni

- `@RestController;`
- `@RequestMapping("/devices/{deviceId}/characteristics").`

Campi privati

- `GetUnarchivedCharacteristicsUseCase getUnarchivedCharacteristicsUseCase`: l'istanza di un servizio a cui inoltrare le richieste per l'ottenimento dell'elenco delle caratteristiche non archiviate;
- `GetLimitsUseCase getLimitsUseCase`: l'istanza di un servizio a cui inoltrare le richieste per l'ottenimento dei limiti di una caratteristica.

Metodi pubblici

- `@GetMapping("") ResponseEntity<List<TinyCharacteristic>> getUnarchivedCharacteristics(@PathVariable("deviceId") int deviceId: l'identificativo della macchina;) throws BusinessException`
Questo metodo implementa l'endpoint descritto in §3.4.2.1.6;
- `@GetMapping("{characteristicId}/limits") ResponseEntity<CharacteristicLimits> getCharacteristicLimits(@PathVariable("deviceId") int deviceId: l'identificativo della macchina; @PathVariable("characteristicId") int characteristicId: l'identificativo della caratteristica, relativo alla macchina;) throws BusinessException`
Questo metodo implementa l'endpoint descritto in §3.4.2.1.8.

4.2.1.7 DetectionsController

Questa classe è un controller di Spring Boot. Il suo ruolo è essere il punto d'entrata delle richieste HTTP effettuate, relative alle rilevazioni di una caratteristica e inoltrarle ai service della parte di business.

Annotazioni

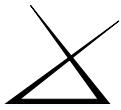
- `@RestController;`
- `@RequestMapping("/devices/{deviceId}/characteristics/{characteristicId}/detections")`.

Campi privati

- `GetDetectionsUseCase getDetectionsUseCase`: l'istanza di un servizio a cui inoltrare le richieste per l'ottenimento della lista delle rilevazioni di una caratteristica.

Metodi pubblici

- `@GetMapping("") DetectionsGroup getCharacteristicDetections(@PathVariable int deviceId: l'identificativo della macchina; @PathVariable int characteristicId: l'identificativo della caratteristica, relativo alla macchina;)`



`@RequestParam(value = "olderThan", required = false) Long olderThan`: il filtro che specifica che devono essere restituite rilevazioni precedenti a un determinato istante. Può non essere specificato;

`@RequestParam(value = "newerThan", required = false) Long newerThan`: il filtro che specifica che devono essere restituite rilevazioni successive a un determinato istante. Può non essere specificato;

`@RequestParam(value = "limit", required = false) Integer limit`: il filtro che specifica la lunghezza massima della lista di rilevazioni ricevuta. Può non essere specificato.

) throws BusinessException

Questo metodo implementa l'endpoint descritto in §3.4.2.1.7.

4.2.1.8 LoginController

Il suo ruolo è essere il punto d'entrata delle richieste HTTP relative all'autenticazione.

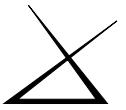
Annotazioni

- `@RestController`;
- `@RequestMapping("/login")`.

Metodi pubblici

- `@GetMapping void login()`

Questo metodo implementa l'endpoint descritto in §3.4.2.1.2. La logica è gestita da Spring Security.



4.2.2 Gestione delle eccezioni

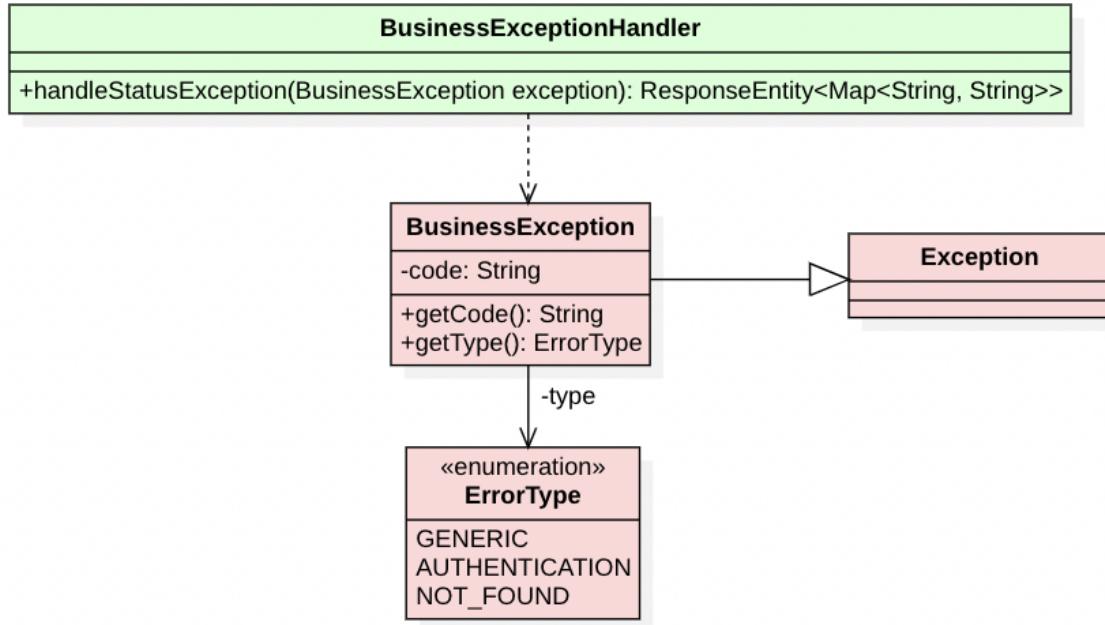


Figura 19: Diagramma delle classi relativo alla gestione delle eccezioni.

Questo modulo si occupa di gestire gli errori dell'utente sotto forma di eccezioni, quindi di raccogliere la motivazione dell'errore e riportarla all'utente. Il ruolo principale è svolto da un `ExceptionHandler` di Spring Boot che cattura le eccezioni lanciate dal `RestController` e risponde di conseguenza.

4.2.2.1 BusinessExceptionHandler

Questa classe si occupa di catturare e gestire le eccezioni lanciate dal controller e produrre un adeguato messaggio di errore per il chiamante.

Annotazioni

- `@ControllerAdvice`.

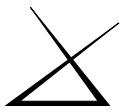
Metodi

- `@ExceptionHandler(BusinessException.class)`
`ResponseEntity<Map<String, String>> handleStatusException(`

`BusinessException exception:` l'eccezione lanciata dal controller.

)

Questo metodo viene chiamato quando il controller lancia un'eccezione. Una volta ricevuta, si occupa di convertirla in un messaggio di errore per il client, con annesso codice di stato HTTP.



4.2.2.2 BusinessException

Questa classe rappresenta un’eccezione derivante dalle componenti di business. Rappresenta un errore causato dall’esterno, e quindi gestibile.

Classe estesa

Questa classe estende `java.lang.Exception`.

Campi privati

- `String code`: il codice di errore;
- `ErrorType type`: il tipo o categoria di errore.

Metodi

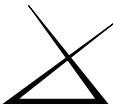
- `String getCode()`
Getter, che ritorna il codice di errore;
- `ErrorType getType()`
Getter che ritorna il tipo di errore.

4.2.2.3 ErrorType

Questa enumerazione rappresenta le possibili tipologie di errore.

Costanti

- `AUTHENTICATION`: caratterizza gli errori di autenticazione;
- `NOT_FOUND`: caratterizza gli errori dovuti a macchine o caratteristiche non esistenti;
- `GENERIC`: caratterizza gli errori generici.



4.2.3 Login

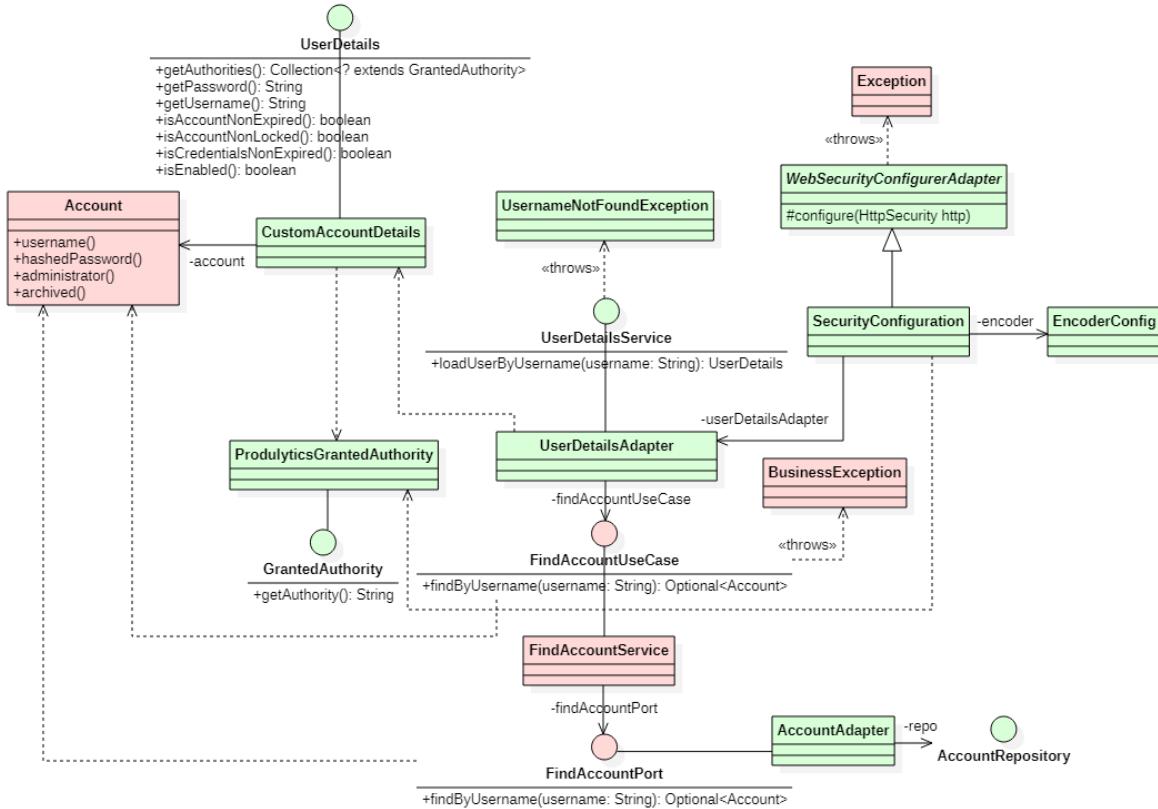


Figura 20: Diagramma delle classi relativo all'autenticazione.

4.2.3.1 CustomAccountDetails

Questa classe implementa la rappresentazione di un utente in Spring Security.

Interfacce implementate

- **UserDetails** di Spring Security.

Campi privati

- **Account account**: un record rappresentante un utente.

Metodi pubblici

- **Collection<ProdulyticsGrantedAuthority> getAuthorities()**
Implementazione dell'omonimo metodo dell'interfaccia **UserDetails**;
- **String getPassword()**
Implementazione dell'omonimo metodo dell'interfaccia **UserDetails**;

- `String getUsername()`
Implementazione dell'omonimo metodo dell'interfaccia `UserDetails`;
- `boolean isAccountNonExpired()`
Implementazione dell'omonimo metodo dell'interfaccia `UserDetails`;
- `boolean isAccountNonLocked()`
Implementazione dell'omonimo metodo dell'interfaccia `UserDetails`;
- `boolean isCredentialsNonExpired()`
Implementazione dell'omonimo metodo dell'interfaccia `UserDetails`;
- `boolean isEnabled()`
Implementazione dell'omonimo metodo dell'interfaccia `UserDetails`.

4.2.3.2 Account

Questo record rappresenta le informazioni di un utente.

Campi privati

- `String username`: l'username dell'utente;
- `String hashedPassword`: la password cifrata dell'utente;
- `boolean administrator`: i permessi dell'utente;
- `boolean archived`: lo stato di archiviazione dell'utente.

4.2.3.3 ProdulyticsGrantedAuthority

Questa enumerazione rappresenta i livelli di permessi esistenti in Produlytics, ossia:

- ADMIN: amministratore;
- ACCOUNT: utente.

Interfacce implementate

- `GrantedAuthority` di Spring Security.

Metodi pubblici

- `String getAuthority()`
Implementazione dell'omonimo metodo dell'interfaccia `GrantedAuthority`.

4.2.3.4 UserDetailsAdapter

Questa classe adatta l'interfaccia `UserDetailsService` di Spring Security.

Annotazioni

- `@Component`.

Interfacce implementate

- `UserDetailsService` di Spring Security.

Campi privati

- `FindAccountUseCase findAccountUseCase`: istanza dell’interfaccia che modella il caso d’uso di ricerca di un utente.

Metodi pubblici

- `UserDetails loadUserByUsername()`

`String username`: l’username dell’utente.

)

Implementazione dell’omonimo metodo dell’interfaccia `UserDetailsService`.

4.2.3.5 FindAccountUseCase

Questa interfaccia modella il caso d’uso che si occupa della ricerca di un utente.

Metodi

- `Optional<Account> findByUsername()`

`String username`: l’username dell’utente da cercare.

)

Verifica che esista un utente con l’username specificato e, in caso positivo, lo restituisce. Lancia un’eccezione se l’utente non è stato trovato.

4.2.3.6 BusinessException

Vedi §4.2.2.2.

4.2.3.7 FindAccountService

Questa classe implementa la ricerca di un utente.

Interfacce implementate

- `FindAccountUseCase`.

Campi privati

- `FindAccountPort findAccountPort`: l’istanza di una porta a cui inoltrare le richieste per la ricerca di un utente dato l’username.

Metodi pubblici

- `Optional<Account> findByUsername()`

`String username`: l’username dell’utente da cercare.

)

Implementazione dell’omonimo metodo dell’interfaccia `FindAccountUseCase`.

4.2.3.8 FindAccountPort

Questa interfaccia rappresenta la porta per la ricerca di un utente all’interno dello strato di persistenza.

Metodi

- `Optional<Account> findByUsername(`

`String username`: l'username dell'utente da cercare.

)

Cerca l'utente e, se lo trova, lo restituisce; altrimenti restituisce `Optional` vuoto.

4.2.3.9 AccountAdapter

Questa classe si occupa di adattare la repository di Spring alle porte descritte dalla parte di business per le operazioni relative agli utenti.

Annotazioni

- `@Component`.

Interfacce implementate

- `FindAccountPort`;
- `UpdateAccountPasswordPort`.

Campi privati

- `AccountRepository repo`: un'istanza del repository degli utenti.

Metodi pubblici

- `void findByUsername(`

`String username`: l'username dell'utente da cercare.

)

Implementazione dell'omonimo metodo definito in `FindAccountPort`;

- `void updateAccountPassword(`

`Account account`: l'utente con la password aggiornata.

)

Implementazione dell'omonimo metodo definito in `UpdateAccountPasswordPort`.

4.2.3.10 AccountRepository

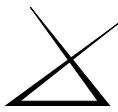
Questa interfaccia espone le query che è possibile effettuare sul database, in particolare sulla tabella `account`, lasciando a Spring il compito di implementarla.

Annotazioni

`@Repository`.

Interfacce estese

`CrudRepository<AccountEntity, String>`.



4.2.3.11 SecurityConfiguration

Questa classe configura Spring Security.

Annotazioni

- `@Configuration;`
- `@EnableWebSecurity.`

Classi astratte estese

- `WebSecurityConfigurerAdapter` di Spring Security.

Campi privati

- `UserDetailsAdapter userDetailsAdapter`: istanza dell'adapter per verificare l'esistenza di un utente;
- `EncoderConfig encoder`: l'istanza della classe che configura il cifratore.

Metodi pubblici

- `@Bean
AuthenticationProvider authenticationProvider()`
Restituisce il provider per l'autenticazione.

Metodi protetti

- `void configure(
HttpSecurity http)`
oggetto per configurare l'autenticazione di Spring Boot.
`) throws Exception`
Override dell'omonimo metodo della classe astratta `WebSecurityConfigurerAdapter`.

4.2.3.12 EncoderConfig

Vedi §4.2.4.10.

4.2.3.13 LoginController

Vedi §4.2.1.8.

4.2.4 Modifica password

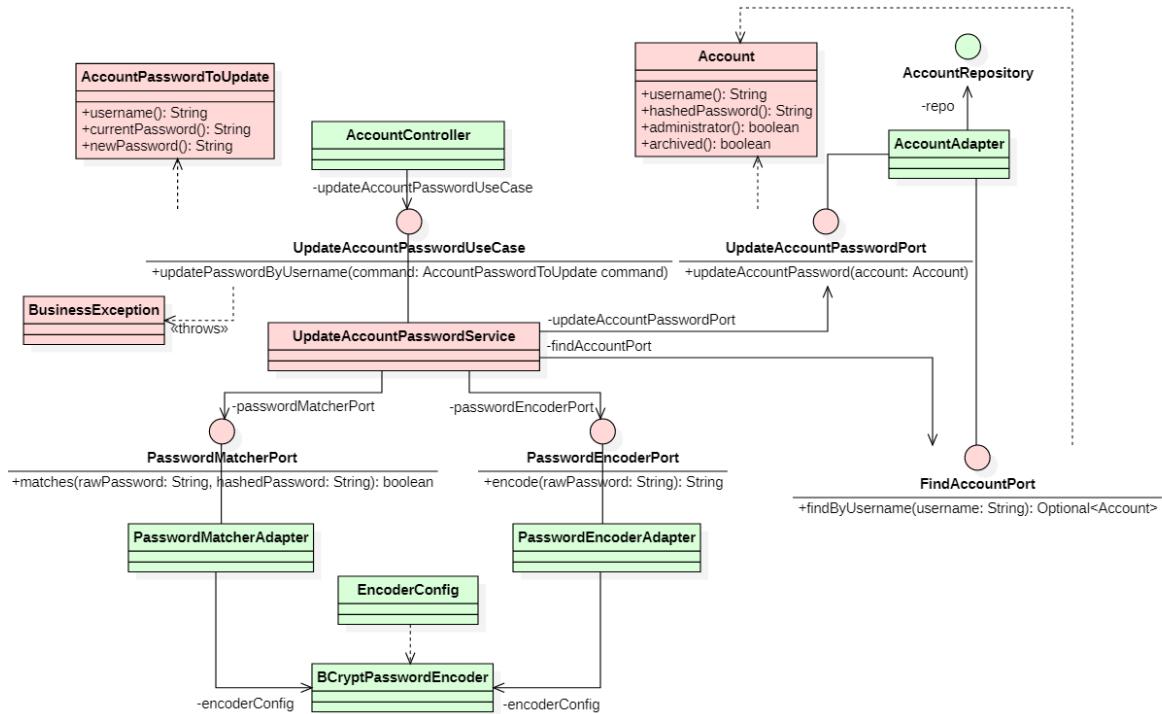


Figura 21: Diagramma delle classi relativo alla modifica della password di un utente.

4.2.4.1 AccountController

Vedi §4.2.1.1.

4.2.4.2 UpdateAccountPasswordUseCase

Questa interfaccia modella il caso d'uso che si occupa della modifica della password di un utente.

Metodi

- `void updatePasswordByUsername(`
- `AccountPasswordToUpdate accountToUpdate: l'utente con i dati per la modifica della password.`
- `) throws BusinessException`
- Verifica che la nuova password soddisfi i requisiti di ammissibilità, che la password attuale inserita sia corretta e, in caso positivo, modifica la password. Lancia un'eccezione se la password non è valida o la password attuale è sbagliata.

4.2.4.3 AccountPasswordToUpdate

Questo record rappresenta un utente con le informazioni per la modifica della password.

Campi privati

- `String username`: l'username dell'utente;
- `String currentPassword`: la password che l'utente ha inserito come corrente e di cui deve essere verificata la correttezza;
- `String newPassword`: la nuova password che l'utente desidera impostare.

4.2.4.4 BusinessException

Vedi §4.2.2.2.

4.2.4.5 UpdateAccountPasswordService

Questa classe implementa la modifica della password di un utente.

Interfacce implementate

- `UpdateAccountPasswordUseCase`.

Campi privati

- `FindAccountPort findAccountPort`: l'istanza di una porta a cui inoltrare le richieste per la ricerca di un utente dato l'username;
- `PasswordMatcherPort passwordMatcherPort`: l'istanza di una porta a cui inoltrare le richieste per il confronto di due password, una in chiaro e una cifrata;
- `PasswordEncoderPort passwordEncoderPort`: l'istanza di una porta a cui inoltrare le richieste per la cifratura di una password in chiaro;
- `UpdateAccountPasswordPort updateAccountPasswordPort`: l'istanza di una porta a cui inoltrare le richieste per l'aggiornamento della password dell'utente.

Metodi pubblici

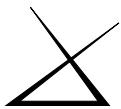
- `void updatePasswordByUsername(`
 `AccountPasswordToUpdate accountToUpdate`: l'utente con i dati per la modifica della password.
 `) throws BusinessException`
Implementazione dell'omonimo metodo dell'interfaccia `UpdateAccountByUsernameUseCase`.

4.2.4.6 PasswordMatcherPort

Questa interfaccia rappresenta la porta che si occupa di stabilire se due password, una in chiaro e una cifrata, corrispondono.

Metodi

- `boolean matches(`
 `String rawPassword`: la password in chiaro;
 `String hashedPassword`: la password cifrata.
 `)`
Confronta le due password e restituisce se corrispondono o meno.



4.2.4.7 PasswordMatcherAdapter

Questa classe si occupa di adattare la classe `BCryptPasswordEncoder` di Spring alla porta `PasswordMatcherPort`.

Annotazioni

- `@Component`.

Interfacce implementate

- `PasswordMatcherPort`.

Campi privati

- `BCryptPasswordEncoder encoderConfig`: il cifratore.

Metodi pubblici

- `String matches()`

```
        String rawPassword: la password in chiaro;
        String hashedPassword: la password cifrata.
    
```

Implementazione dell'omonimo metodo dell'interfaccia `PasswordMatcherPort`.

4.2.4.8 PasswordEncoderPort

Questa interfaccia rappresenta la porta che si occupa della cifratura di una password in chiaro.

Metodi

- `String encode()`

```
        String rawPassword: la password in chiaro da cifrare.
    
```

Cifra la password e la restituisce.

4.2.4.9 PasswordEncoderAdapter

Questa classe si occupa di adattare la classe `BCryptPasswordEncoder` di Spring alla porta `PasswordEncoderPort`.

Annotazioni

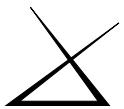
- `@Component`.

Interfacce implementate

- `PasswordEncoderPort`.

Campi privati

- `BCryptPasswordEncoder encoderConfig`: il cifratore.



Metodi pubblici

- `public String encode(`

`String rawPassword:` la password in chiaro da cifrare.

)

Implementazione dell'omonimo metodo dell'interfaccia `PasswordEncoderPort`.

4.2.4.10 EncoderConfig

Questa classe rappresenta la configurazione del cfratore.

Annotazioni

- `Configuration`.

Metodi

- `@Bean`

`BCryptPasswordEncoder getEncoder()`

Restituisce il cfratore.

4.2.4.11 UpdateAccountPasswordPort

Questa interfaccia rappresenta la porta per l'aggiornamento della password di un utente.

Metodi

- `void updateAccountPassword(`

`Account account:` l'utente con le informazioni aggiornate.

)

Memorizza l'utente con le informazioni aggiornate.

4.2.4.12 FindAccountPort

Vedi §4.2.3.8.

4.2.4.13 Account

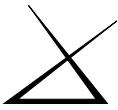
Vedi §4.2.3.2.

4.2.4.14 AccountAdapter

Vedi §4.2.3.9.

4.2.4.15 AccountRepository

Vedi §4.2.3.10.



4.2.5 Lista macchine non archiviate

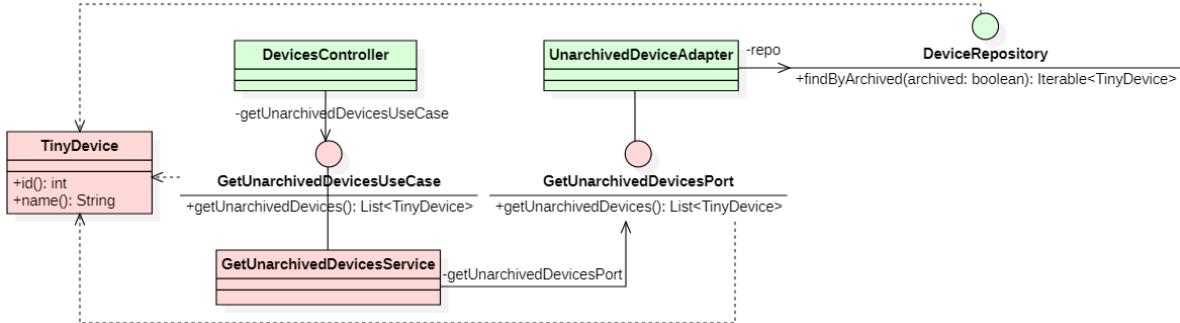


Figura 22: Diagramma delle classi relativo all'ottenimento della lista delle macchine non archiviate.

4.2.5.1 DevicesController

Vedi §4.2.1.4.

4.2.5.2 GetUnarchivedDevicesUseCase

Questa interfaccia modella il caso d'uso che si occupa dell'ottenimento della lista delle macchine non archiviate.

Metodi

- `List<TinyDevice> getUnarchivedDevices()`
Restituisce la lista delle macchine non archiviate.

4.2.5.3 TinyDevice

Questo record rappresenta l'intestazione di una macchina, con l'identificativo e il nome.

Campi privati

- `int id`: l'identificativo della macchina;
- `String name`: il nome della macchina.

4.2.5.4 GetUnarchivedDevicesService

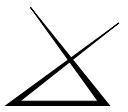
Questa classe implementa l'ottenimento della lista delle macchine non archiviate.

Interfacce implementate

- `GetUnarchivedDevicesUseCase`.

Campi privati

- `GetUnarchivedDevicesPort getUnarchivedDevicesPort`: l'istanza della porta a cui inoltrare le richieste per l'ottenimento della lista delle macchine non archiviate.



Metodi pubblici

- `List<TinyDevice> getUnarchivedDevices()`
Implementazione dell'omonimo metodo dell'interfaccia `GetUnarchivedDevicesUseCase`.

4.2.5.5 GetUnarchivedDevicesPort

Questa interfaccia modella la porta per l'ottenimento della lista delle macchine non archiviate.

Metodi

- `List<TinyDevice> getUnarchivedDevices()`.

4.2.5.6 UnarchivedDeviceAdapter

Questa classe si occupa di adattare il repository di Spring alle porte descritte dalla parte di business per le operazioni relative alle macchine non archiviate.

Annotazioni

- `@Component`.

Interfacce implementate

- `GetUnarchivedDevicesPort`.

Campi privati

- `DeviceRepository repo`: un'istanza del repository delle macchine.

Metodi pubblici

- `List<TinyDevice> getUnarchivedDevices()`
Implementazione dell'omonimo metodo definito in `GetUnarchivedDevicesPort`.

4.2.5.7 DeviceRepository

Questa interfaccia espone le query che è possibile effettuare sul database, in particolare sulla tabella `device`, lasciando a Spring il compito di implementarla.

Annotazioni

- `@Repository`.

Interfacce estese

- `JpaRepository<DeviceEntity, String>`.

Metodi

- `Iterable<TinyDevice> findByArchived(
 boolean archived)`
 `boolean archived`: lo stato di archiviazione della macchina.

 `)`
 Restituisce le macchine con stato di archiviazione pari a `archived`.

4.2.6 Lista caratteristiche non archiviate di una macchina

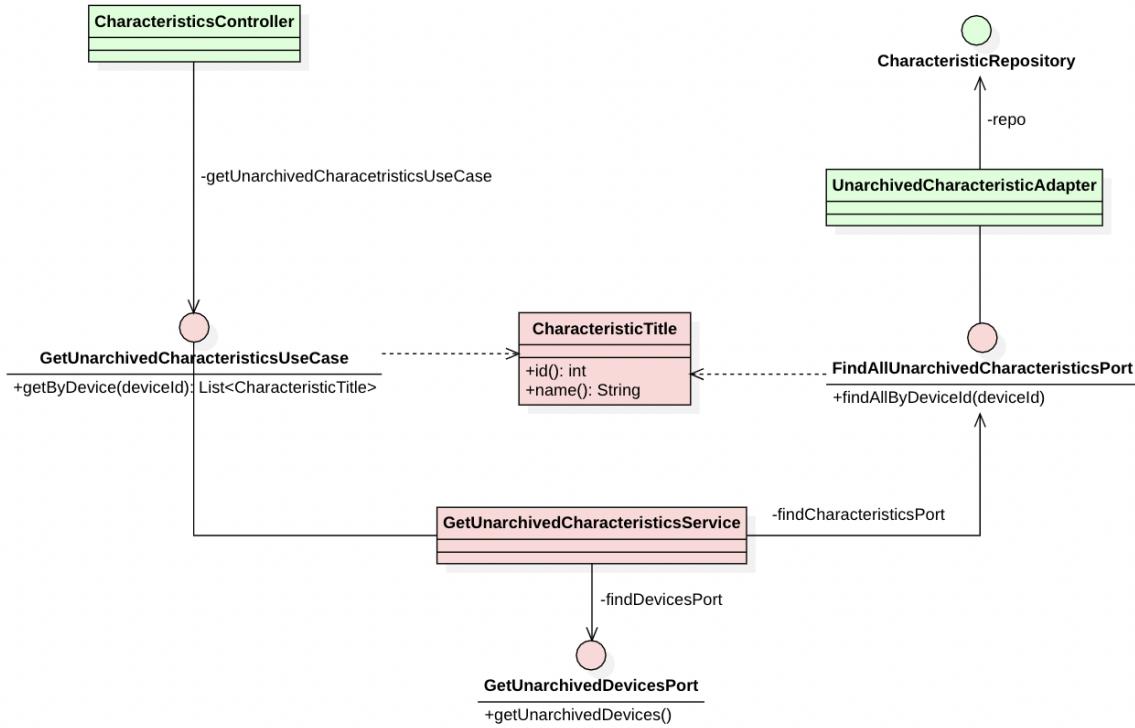


Figura 23: Diagramma delle classi relativo all’ottenimento della lista delle caratteristiche non archiviate di una macchina.

4.2.6.1 CharacteristicsController

Vedi §4.2.1.6.

4.2.6.2 GetUnarchivedCharacteristicsUseCase

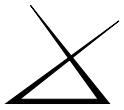
Questa interfaccia modella il caso d’uso che si occupa dell’ottenimento della lista delle caratteristiche non archiviate di una macchina non archiviata.

Metodi

- `List<TinyCharacteristic> getByDevice(`
- `int deviceId: l’identificativo della macchina.`
- `) throws BusinessException`
- `Restituisce la lista delle caratteristiche non archiviate della macchina.`

4.2.6.3 BusinessException

Vedi §4.2.2.2.



4.2.6.4 TinyCharacteristic

Questo record rappresenta l'intestazione di una caratteristica, con l'identificativo e il nome.

Campi privati

- int id: l'identificativo della caratteristica, relativo alla macchina;
- String name: il nome della caratteristica.

4.2.6.5 GetUnarchivedCharacteristicsService

Questa classe implementa l'ottenimento della lista delle caratteristiche non archiviate di una macchina non archiviata.

Interfacce implementate

- GetUnarchivedCharacteristicsUseCase.

Campi privati

- GetUnarchivedDevicesPort findDevicesPort: l'istanza della porta a cui inoltrare le richieste per l'ottenimento della lista delle macchine non archiviate;
- FindAllUnarchivedCharacteristicsPort findCharacteristicsPort: l'istanza della porta a cui inoltrare le richieste per l'ottenimento di tutte le caratteristiche non archiviate di una macchina.

Metodi pubblici

- List<TinyCharacteristic> getByDevice(
 int deviceId: l'identificativo della macchina.
)
throws BusinessException
Implementazione dell'omonimo metodo dell'interfaccia GetUnarchivedCharacteristicsUseCase.

4.2.6.6 GetUnarchivedDevicesPort

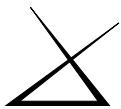
Vedi §4.2.5.5.

4.2.6.7 FindAllUnarchivedCharacteristicsPort

Questa interfaccia modella la porta per l'ottenimento della lista delle caratteristiche non archiviate di una macchina.

Metodi

- List<TinyCharacteristic> findAllByDeviceId(
 int deviceId: l'identificativo della macchina.
)
Cerca le caratteristiche non archiviate della macchina con l'identificativo dato e le restituisce.



4.2.6.8 UnarchivedCharacteristicAdapter

Questa classe si occupa di adattare il repository di Spring alle porte descritte dalla parte di business per le operazioni relative alle caratteristiche.

Annotazioni

- `@Component`.

Interfacce implementate

- `FindAllUnarchivedCharacteristicsPort`;
- `FindCharacteristicLimitsPort`.

Campi privati

- `CharacteristicRepository repo`: un'istanza del repository delle caratteristiche.

Metodi pubblici

- ```
List<TinyCharacteristic> findAllByDeviceId(
 int deviceId: l'identificativo della macchina.
)
Implementazione dell'omonimo metodo definito in FindAllUnarchivedCharacteristicsPort;
```
- ```
Optional<CharacteristicLimits> findByCharacteristic(  
    int deviceId: l'identificativo della macchina;  
    int characteristicId: l'identificativo della caratteristica, relativo alla macchina.  
)  
Implementazione dell'omonimo metodo definito in FindCharacteristicLimitsPort.
```

4.2.6.9 CharacteristicRepository

Questa interfaccia espone le query che è possibile effettuare sul database, in particolare sulla tabella `characteristic`, lasciando a Spring il compito di implementarla.

Annotazioni

- `@Repository`.

Interfacce estese

- `JpaRepository<CharacteristicEntity, CharacteristicEntityId>`.

Metodi

- ```
List<CharacteristicEntity> findByArchivedFalseAndDeviceId(
 int deviceId: l'identificativo della macchina.
)
```

#### 4.2.7 Rilevazioni di una caratteristica

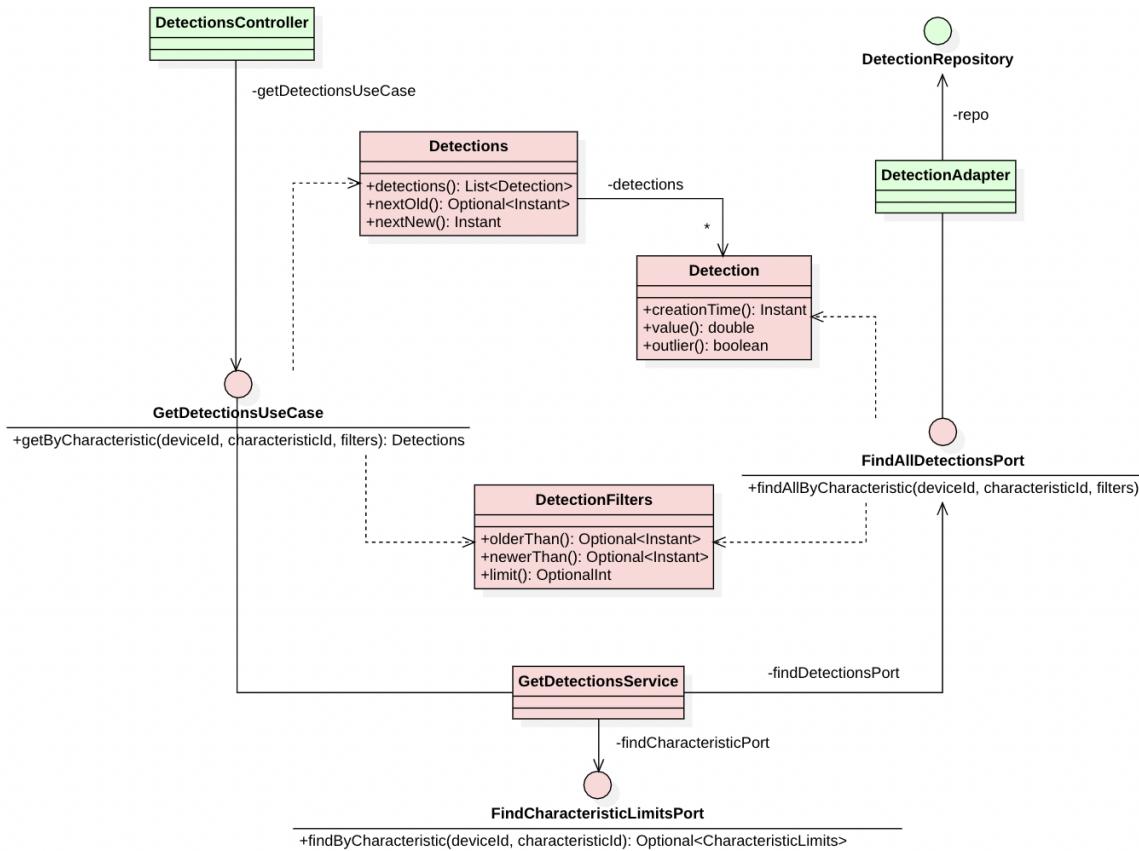


Figura 24: Diagramma delle classi relativo all'ottenimento della lista delle rilevazioni di una caratteristica.

##### 4.2.7.1 DetectionsController

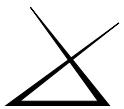
Vedi §4.2.1.7

##### 4.2.7.2 DetectionFilters

Questo record rappresenta i filtri applicabili alla ricerca di una lista di rilevazioni.

##### Campi privati

- `OptionalLong olderThan`: il filtro che specifica che devono essere restituite rilevazioni precedenti a un determinato istante. Può non essere specificato;
- `OptionalLong newerThan`: il filtro che specifica che devono essere restituite rilevazioni successive a un determinato istante. Può non essere specificato;
- `OptionalInt limit`: il filtro che specifica la lunghezza massima della lista di rilevazioni ricevuta. Può non essere specificato.



### Metodi statici

- `DetectionFiltersBuilder builder()`

Fornisce il builder del record con nessun valore specificato.

#### 4.2.7.3 GetDetectionsUseCase

Questa interfaccia modella il caso d'uso che si occupa dell'ottenimento della lista delle rilevazioni di una caratteristica non archiviata.

### Metodi

- `DetectionsGroup listByCharacteristic(`

`int deviceId:` l'identificativo della macchina;

`int characteristicId:` l'identificativo della caratteristica, relativo alla macchina;

`DetectionFilters filters:` i filtri di ricerca richiesti.

`) throws BusinessException`

Restituisce la lista delle rilevazioni trovate della caratteristica non archiviata, applicando i filtri richiesti.

#### 4.2.7.4 DetectionsGroup

Questo record rappresenta una lista di rilevazioni.

### Campi privati

- `List<Detection> detections:` la lista di rilevazioni;
- `OptionalLong nextOld:` il timestamp per ottenere le liste delle rilevazioni più vecchie in una nuova richiesta;
- `long nextNew:` il timestamp per ottenere le liste delle rilevazioni più recenti in una nuova richiesta.

#### 4.2.7.5 Detection

Questo record rappresenta tutte le informazioni di una rilevazione

### Campi privati

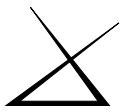
- `long creationTime:` l'istante della rilevazione come millisecondi trascorsi dallo UNIX epoch;
- `double value:` il valore rilevato;
- `boolean outlier:` `true` se la rilevazione è anomala; `false` altrimenti.

#### 4.2.7.6 BusinessException

Vedi §4.2.2.2.

#### 4.2.7.7 GetDetectionsService

Questa classe implementa l'ottenimento della lista delle rilevazioni di una caratteristica non archiviata.



### Interfacce implementate

- `GetDetectionsUseCase.`

### Campi privati

- `FindAllDetectionsPort findDetectionsPort`: l'istanza della porta a cui inoltrare le richieste per l'ottenimento delle rilevazioni di una caratteristica;
- `FindCharacteristicLimitsPort findCharacteristicPort`: l'istanza della porta a cui inoltrare le richieste per l'ottenimento dei limiti di una caratteristica.

### Metodi pubblici

- `DetectionsGroup listByCharacteristic(  
    int deviceId: l'identificativo della macchina;  
    int characteristicId: l'identificativo della caratteristica, relativo alla macchina;  
    DetectionFilters filters: i filtri di ricerca richiesti.  
) throws BusinessException`  
Implementazione dell'omonimo metodo dell'interfaccia `GetDetectionsUseCase`.

#### 4.2.7.8 FindAllDetectionsPort

Questa interfaccia modella la porta per l'ottenimento della lista delle rilevazioni di una caratteristica non archiviata.

### Metodi

- `List<Detection> findAllByCharacteristic(  
    int deviceId: l'identificativo della macchina;  
    int characteristicId: l'identificativo della caratteristica, relativo alla macchina;  
    OptionalLong olderThan: il filtro che specifica che devono essere restituite rilevazioni precedenti a un determinato istante. Può non essere specificato.`

Cerca le rilevazioni della caratteristica specificata, applicando il filtro (se richiesto) e le restituisce.

#### 4.2.7.9 FindCharacteristicLimitsPort

Vedi §4.2.8.7.

#### 4.2.7.10 DetectionAdapter

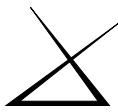
Questa classe si occupa di adattare il repository di Spring alle porte descritte dalla parte di business per le operazioni relative alle rilevazioni delle caratteristiche.

### Annotazioni

- `@Component.`

### Interfacce implementate

- `FindAllDetectionsPort.`



### Campi privati

- `DetectionRepository repo`: un'istanza del repository delle rilevazioni.

### Metodi pubblici

- `List<Detection> findAllByCharacteristic(`  
`int deviceId`: l'identificativo della macchina;  
`int characteristicId`: l'identificativo della caratteristica, relativo alla macchina;  
`OptionalLong olderThan`: il filtro che specifica che devono essere restituite rilevazioni più vecchie di un determinato istante. Può non essere specificato.

Implementazione dell'omonimo metodo definito in `FindAllDetectionsPort`.

#### 4.2.7.11 DetectionsRepository

Questa interfaccia espone le query che è possibile effettuare sul database, in particolare sulla tabella `detections`.

### Annotazioni

- `@Repository`.

### Interfacce estese

- `CrudRepository<DetectionEntity, DetectionEntityId>`.

### Metodi

- `@Query(value = ...)`  
`List<DetectionEntity> findByCharacteristicAndCreationTimeGreaterThanOrQuery(`  
`@Param("deviceId") int deviceId`: l'identificativo della macchina;  
`@Param("characteristicId") int characteristicId`: l'identificativo della caratteristica, relativo alla macchina;  
`@Param("olderThan") Instant olderThan`: il filtro che specifica che devono essere restituite rilevazioni più vecchie di un determinato istante. Può essere `null`.  
)

Restituisce la lista di tutte le rilevazioni della caratteristica di una macchina, eventualmente più vecchie dell'istante specificato.

#### 4.2.8 Limiti e media di una caratteristica

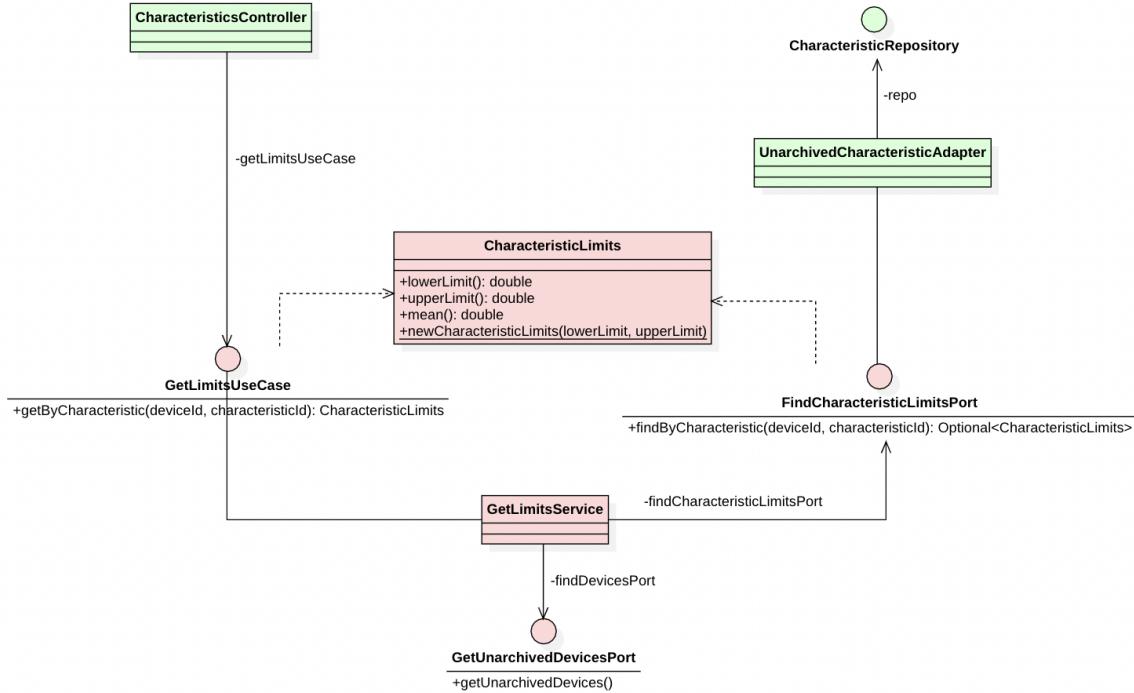


Figura 25: Diagramma delle classi relativo all'ottenimento dei limiti tecnici di una caratteristica.

##### 4.2.8.1 CharacteristicsController

Vedi §4.2.1.6.

##### 4.2.8.2 GetLimitsUseCase

Questa interfaccia modella il caso d'uso che si occupa dell'ottenimento dei limiti tecnici di una caratteristica non archiviata.

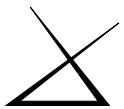
##### Metodi

- `CharacteristicLimits getByCharacteristic(`
- ```

        int deviceId: l'identificativo della macchina;
        int characteristicsId: l'identificativo della caratteristica, relativo alla macchina.
    ) throws BusinessException
    Restituisce i limiti tecnici di una caratteristica non archiviata.
    
```

4.2.8.3 BusinessException

Vedi §4.2.2.2.



4.2.8.4 CharacteristicLimits

Questo record rappresenta i limiti tecnici di una caratteristica e la loro media.

Campi privati

- `double lowerLimit`: il limite tecnico inferiore;
- `double upperLimit`: il limite tecnico superiore;
- `mean`: la media dei limiti.

Metodi pubblici

- `static double mean()`
 - `double lowerLimit`: il limite tecnico inferiore;
 - `double upperLimit`: il limite tecnico superiore.

Restituisce una nuova istanza del record con i limiti specificati e la media calcolata automaticamente.

4.2.8.5 GetLimitsService

Questa classe implementa l'ottenimento dei limiti tecnici di una caratteristica non archiviata.

Interfacce implementate

- `GetLimitsUseCase`.

Campi privati

- `GetUnarchivedDevicesPort findDevicesPort`: l'istanza della porta a cui inoltrare le richieste per l'ottenimento delle macchine non archiviate;
- `FindCharacteristicLimitsPort findCharacteristicLimitsPort`: l'istanza della porta a cui inoltrare le richieste per l'ottenimento dei limiti di una caratteristica.

Metodi pubblici

- `CharacteristicLimits getByCharacteristic(`
 - `int deviceId`: l'identificativo della macchina;
 - `int characteristicId`: l'identificativo della caratteristica, relativo alla macchina.`) throws BusinessException`
Implementazione dell'omonimo metodo dell'interfaccia `GetLimitsUseCase`.

4.2.8.6 GetUnarchivedDevicesPort

Vedi §4.2.5.5.

4.2.8.7 FindCharacteristicLimitsPort

Questa interfaccia modella la porta per l'ottenimento dei limiti di una caratteristica.

Metodi

- `Optional<CharacteristicLimits> findByCharacteristic(`
`int deviceId: l'identificativo della macchina;`
`int characteristicId l'identificativo della caratteristica, relativo alla macchina.`
`)`
- Cerca la caratteristica con l'identificativo dato e ne restituisce i limiti tecnici, se presenti.

4.2.8.8 UnarchivedCharacteristicAdapter

Vedi §4.2.6.8.

4.2.8.9 CharacteristicRepository

Vedi §4.2.6.9.

4.2.9 Lista macchine

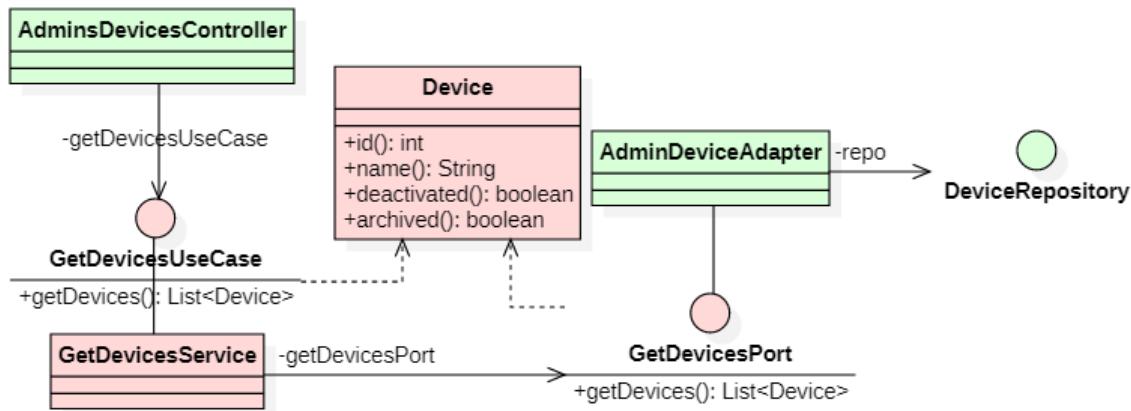


Figura 26: Diagramma delle classi relativo all'ottenimento della lista delle macchine.

4.2.9.1 AdminsDevicesController

Vedi §4.2.1.3.

4.2.9.2 GetDevicesUseCase

Questa interfaccia modella il caso d'uso che si occupa dell'ottenimento della lista delle macchine.

Metodi

- `List<Device> getDevices()`
 Restituisce la lista delle macchine.

4.2.9.3 Device

Questo record rappresenta una macchina.

Campi privati

- `int id`: l'identificativo della macchina;
- `String name`: il nome della macchina;
- `boolean deactivated`: lo stato di attivazione della macchina;
- `boolean archived`: lo stato di archiviazione della macchina.

4.2.9.4 GetDevicesService

Questa classe implementa l'ottenimento della lista delle macchine.

Interfacce implementate

- `GetDevicesUseCase`.

Campi privati

- `GetDevicesPort getDevicesPort`: l'istanza della porta a cui inoltrare le richieste per l'ottenimento della lista delle macchine.

Metodi pubblici

- `List<Device> getDevices()`
Implementazione dell'omonimo metodo dell'interfaccia `GetDevicesUseCase`.

4.2.9.5 GetDevicesPort

Questa interfaccia modella la porta per l'ottenimento di tutte le macchine.

Metodi

- `List<Device> getDevices()`
Ritorna la lista delle macchine presenti.

4.2.9.6 AdminDeviceAdapter

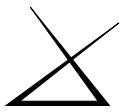
Questa classe si occupa di adattare il repository di Spring alle porte descritte dalla parte di business per le operazioni effettuate dagli amministratori, riguardanti le macchine.

Annotazioni

- `@Component`.

Interfacce implementate

- `InsertDevicePort`;
- `GetDevicesPort`;
- `FindTinyDevicePort`;
- `FindDetailedDevicePort`;
- `UpdateDeviceArchiveStatusPort`;



- `UpdateDeviceDeactivateStatusPort;`
- `GetDeviceDetailsPort;`
- `UpdateDeviceNamePort.`

Campi privati

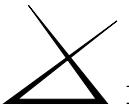
- `DeviceRepository repo:` un'istanza del repository delle macchine.

Metodi pubblici

- `List<Device> getDevices()`
Restituisce la lista delle macchine;
- `Optional<TinyDevice> findTinyDevice(`
`int deviceId:` l'identificativo della macchina.
)
Restituisce una macchina con le sue informazioni essenziali, dato il suo identificativo;
- `Optional<DetailedDevice> findDetailedDevice(`
`int deviceId:` l'identificativo della macchina.
)
Restituisce la macchina dettagliata, dato il suo identificativo;
- `updateDeviceArchiveStatus(`
`DetailedDevice device:` la macchina aggiornata.
)
Aggiorna lo stato di archiviazione di una macchina;
- `updateDeviceDeactivateStatus(`
`DetailedDevice device:` la macchina aggiornata.
)
Aggiorna lo stato di attivazione di una macchina;
- `updateDeviceName(`
`DetailedDevice device:` la macchina aggiornata.
)
Aggiorna il nome di una macchina;
- `insertDevice(`
`NewDevice device:` la nuova macchina.
)
Memorizza una nuova macchina.

4.2.9.7 DeviceRepository

Vedi §4.2.5.7.



4.2.10 Archiviazione macchina

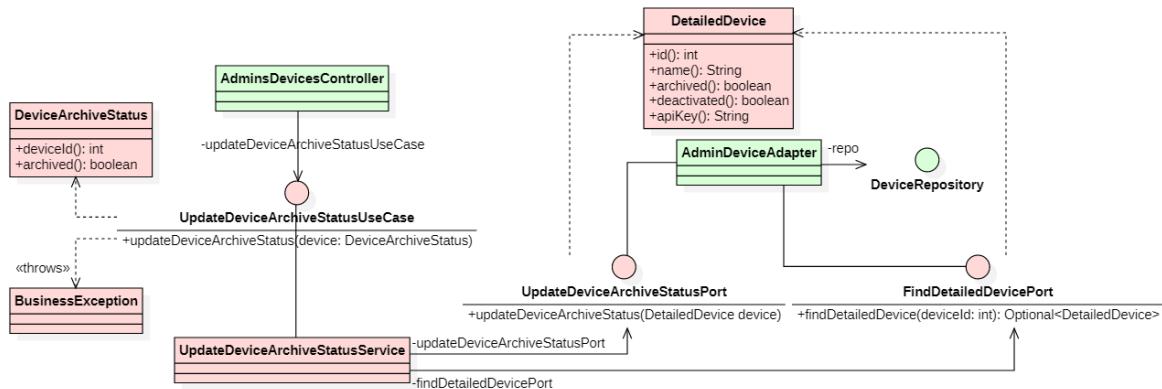


Figura 27: Diagramma delle classi relativo all'aggiornamento dello stato di archiviazione di una macchina.

4.2.10.1 AdminsDevicesController

Vedi §4.2.1.3.

4.2.10.2 UpdateDeviceArchiveStatusUseCase

Questa interfaccia modella il caso d'uso che si occupa della modifica dello stato di archiviazione di una macchina.

Metodi

- `void updateDeviceArchiveStatus(`
 `DeviceArchiveStatus device`: la macchina con lo stato di archiviazione modificato.
 `) throws BusinessException`
 Verifica che la macchina esista e modifica lo stato di archiviazione con il valore dato. Lancia un'eccezione se la macchina non esiste.

4.2.10.3 UpdateDeviceArchiveStatusService

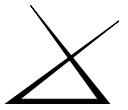
Questa classe implementa la modifica dello stato di archiviazione di una macchina.

Interfacce implementate

- `UpdateDeviceArchiveStatusUseCase`.

Campi privati

- `FindDetailedDevicePort findDetailedDevicePort`: l'istanza della porta a cui inoltrare le richieste per la ricerca di una macchina dato l'identificativo;
- `UpdateDeviceArchiveStatusPort updateDeviceArchiveStatusPort`: l'istanza della porta a cui inoltrare le richieste per l'aggiornamento dello stato di archiviazione di una macchina.



Metodi pubblici

- `void updateDeviceArchiveStatus(`
 `DeviceArchiveStatus device`: la macchina con lo stato di archiviazione modificato.
 `) throws BusinessException`
 Implementazione dell'omonimo metodo dell'interfaccia `UpdateDeviceArchiveStatusUseCase`.

4.2.10.4 DeviceArchiveStatus

Questo record rappresenta una macchina con lo stato di archiviazione modificato da un amministratore.

Campi privati

- `int id`: l'identificativo della macchina;
- `boolean archived`: lo stato di archiviazione da impostare alla macchina.

4.2.10.5 FindDetailedDevicePort

Questa interfaccia modella la porta per la ricerca di una macchina, completa di tutte le sue informazioni.

Metodi

- `Optional<DetailedDevice> findDetailedDevice(`
 `int deviceId`: l'identificativo della macchina da cercare.
 `)`
 Ritorna la macchina con tutte le sue informazioni, se presente; `Optional` vuoto, altrimenti.

4.2.10.6 DetailedDevice

Questo record rappresenta una macchina completa di tutte le sue informazioni.

Campi privati

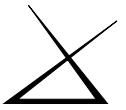
- `int id`: l'identificativo della macchina;
- `String name`: il nome della macchina;
- `boolean deactivated`: lo stato di attivazione della macchina;
- `boolean archived`: lo stato di archiviazione della macchina;
- `String apiKey`: la chiave della API rilevazioni.

4.2.10.7 UpdateDeviceArchiveStatusPort

Questa interfaccia modella la porta per la modifica dello stato di archiviazione di una macchina.

Metodi

- `void updateDeviceArchiveStatus(`
 `DetailedDevice device`: la macchina con lo stato di archiviazione modificato.
 `)`
 Aggiorna lo stato di archiviazione della macchina.



4.2.10.8 AdminDeviceAdapter

Vedi §4.2.9.6.

4.2.10.9 DeviceRepository

Vedi §4.2.5.7.

4.2.11 Disattivazione macchina

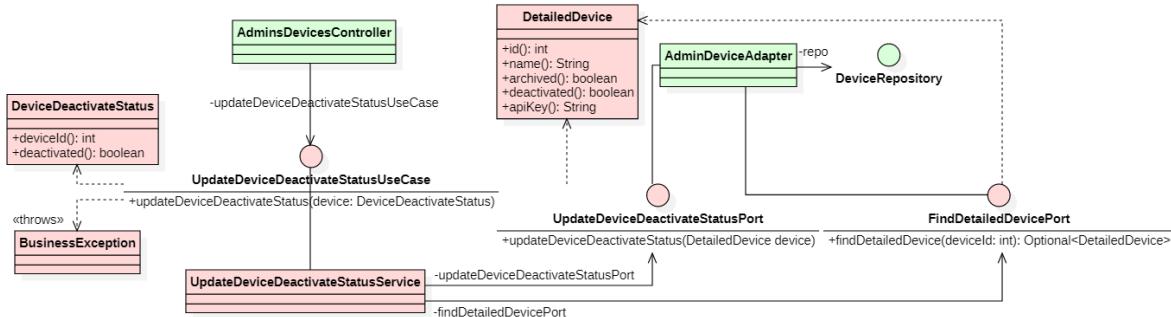


Figura 28: Diagramma delle classi relativo all'aggiornamento dello stato di attivazione di una macchina.

4.2.11.1 AdminsDevicesController

Vedi §4.2.1.3.

4.2.11.2 UpdateDeviceDeactivateStatusUseCase

Questa interfaccia modella il caso d'uso che si occupa della modifica dello stato di attivazione di una macchina.

Metodi

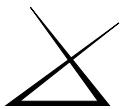
- `void updateDeviceDeactivateStatus(`
 `DeviceDeactivateStatus device:` la macchina con lo stato di attivazione modificato.
 `) throws BusinessException`
 Verifica che la macchina esista e modifica lo stato di attivazione con il valore assegnato. Lancia un'eccezione se la macchina non esiste.

4.2.11.3 DeviceDeactivateStatus

Questo record rappresenta una macchina con stato di attivazione modificato da un amministratore.

Campi privati

- `int id:` l'identificativo della macchina;
- `boolean deactivated:` lo stato di attivazione da impostare alla macchina.



4.2.11.4 BusinessException

Vedi §4.2.2.2.

4.2.11.5 UpdateDeviceDeactivateStatusService

Questa classe implementa la modifica dello stato di attivazione di una macchina.

Interfacce implementate

- `UpdateDeviceDeactivateStatusUseCase`.

Campi privati

- `FindDetailedDevicePort findDetailedDevicePort`: l'istanza della porta a cui inoltrare le richieste per la ricerca di una macchina dato l'identificativo;
- `UpdateDeviceDeactivateStatusPort updateDeviceDeactivateStatusPort`: l'istanza della porta a cui inoltrare le richieste per l'aggiornamento dello stato di attivazione di una macchina.

Metodi pubblici

- `void updateDeviceDeactivateStatus(
 DeviceDeactivateStatus device
) throws BusinessException`
Implementazione dell'omonimo metodo dell'interfaccia `UpdateDeviceDeactivateStatusUseCase`.

4.2.11.6 FindDetailedDevicePort

Vedi §4.2.10.5.

4.2.11.7 UpdateDeviceDeactivateStatusPort

Questa interfaccia modella la porta per la modifica dello stato di attivazione di una macchina.

Metodi

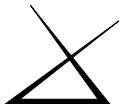
- `void updateDeviceDeactivateStatus(
 DetailedDevice device
)
 Aggiorna lo stato di attivazione della macchina.`

4.2.11.8 AdminDeviceAdapter

Vedi §4.2.9.6.

4.2.11.9 DeviceRepository

Vedi §4.2.5.7.



4.2.12 Dettagli macchina

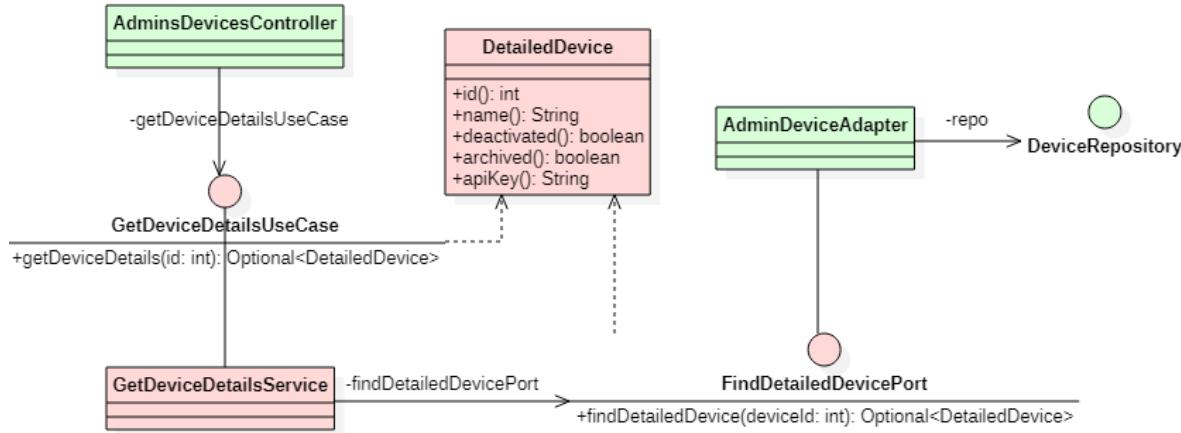


Figura 29: Diagramma delle classi relativo all'ottenimento dei dettagli di una macchina.

4.2.12.1 AdminsDevicesController

Vedi §4.2.1.3.

4.2.12.2 GetDeviceDetailsUseCase

Questa interfaccia modella il caso d'uso che si occupa dell'ottenimento dei dettagli di una macchina.

Metodi

- `DetailedDevice getDeviceDetails(`
 `int id: l'identificativo della macchina.`
 `)`
 `Restituisce i dettagli della macchina.`

4.2.12.3 DetailedDevice

Vedi §4.2.10.6

4.2.12.4 GetDeviceDetailsService

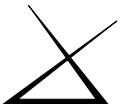
Questa classe implementa l'ottenimento dei dettagli di una macchina.

Interfacce implementate

- GetDeviceDetailsUseCase.

Campi privati

- **FindDetailedDevicePort** `findDetailedDevicePort`: l'istanza della porta a cui inoltrare le richieste per l'ottenimento dei dettagli di una macchina.



Metodi pubblici

- Detailed Device getDeviceDetails()
 int id: l'identificativo della macchina.
)

Implementazione dell'omonimo metodo dell'interfaccia `GetDeviceDetailsUseCase`.

4.2.12.5 FindDetailedDevicePort

Vedi §4.2.10.5.

4.2.12.6 AdminDeviceAdapter

Vedi §4.2.9.6.

4.2.12.7 DeviceRepository

Vedi §4.2.5.7.

4.2.13 Modifica macchina

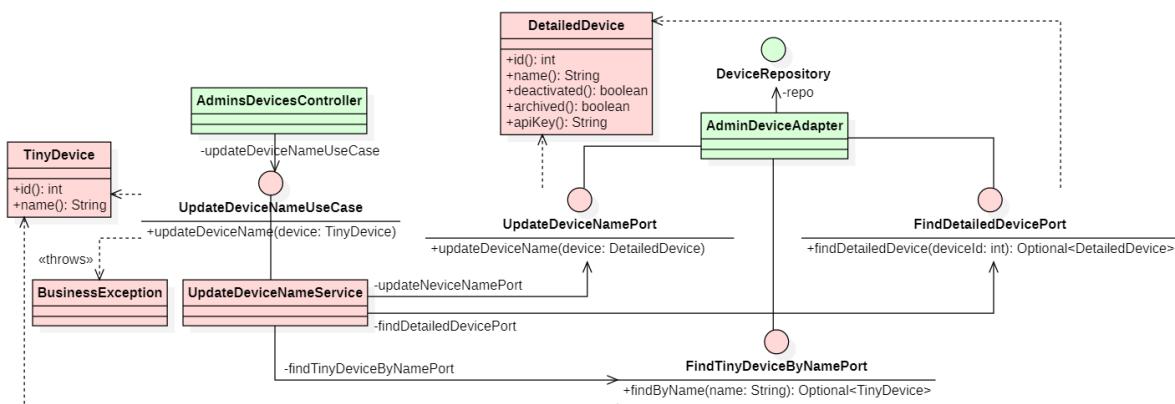


Figura 30: Diagramma delle classi relativo alla modifica di una macchina.

4.2.13.1 AdminsDevicesController

Vedi §4.2.1.3.

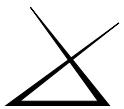
4.2.13.2 UpdateDeviceNameUseCase

Questa interfaccia modella il caso d'uso che si occupa della modifica del nome di una macchina.

Metodi

- void updateDeviceName(

`TinyDevice device`: la macchina con il nome modificato.



```
) throws BusinessException
```

Verifica che la macchina esista, che il nuovo modo da impostare non sia già assegnato a un'altra macchina e, in caso affermativo, effettua la chiamata per l'aggiornamento del suo nome. Lancia un'eccezione se la macchina non è stata trovata o il nome è già usato.

4.2.13.3 TinyDevice

Vedi §4.2.5.3.

4.2.13.4 BusinessException

Vedi §4.2.2.2.

4.2.13.5 UpdateDeviceNameService

Questa classe implementa la modifica del nome di una macchina.

Interfacce implementate

- UpdateDeviceNameUseCase.

Campi privati

- FindDetailedDevicePort findDetailedDevicePort: l'istanza di una porta a cui inoltrare le richieste per la ricerca di una macchina dato l'identificativo;
- FindTinyDeviceByNamePort findTinyDeviceByNamePort: l'istanza di una porta a cui inoltrare le richieste per la ricerca di una macchina dato il nome;
- UpdateDeviceNamePort updateDeviceNamePort: l'istanza di una porta a cui inoltrare le richieste per l'aggiornamento del nome di una macchina.

Metodi pubblici

- void updateDeviceName(

TinyDevice device: la macchina con il nome modificato.

```
) throws BusinessException
```

Implementazione dell'omonimo metodo dell'interfaccia UpdateDeviceNameUseCase.

4.2.13.6 FindDetailedDevicePort

Vedi §4.2.10.5.

4.2.13.7 DetailedDevice

Vedi §4.2.10.6

4.2.13.8 FindTinyDeviceByNamePort

Questa interfaccia rappresenta la porta per la ricerca di una macchina con le informazioni essenziali, dato il suo nome.

Metodi

- `Optional<TinyDevice> findByName(`

`String name:` il nome della macchina da cercare.

)

Ritorna la macchina con le informazioni essenziali, se presente.

4.2.13.9 UpdateDeviceNamePort

Questa interfaccia rappresenta la porta per l'aggiornamento del nome di una macchina.

Metodi

- `void updateDeviceName(`

`DetailedDevice device:` la macchina con le informazioni aggiornate.

)

Memorizza il nome aggiornato della macchina.

4.2.13.10 AdminDeviceAdapter

Vedi §4.2.9.6.

4.2.13.11 DeviceRepository

Vedi §4.2.5.7.

4.2.14 Inserimento macchina

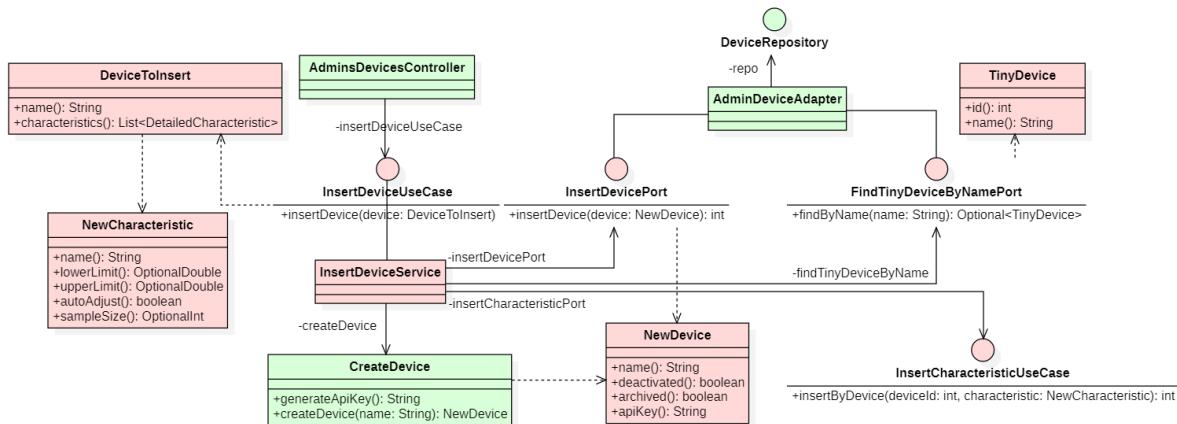
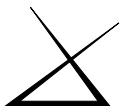


Figura 31: Diagramma delle classi relativo all'inserimento di una nuova macchina.

4.2.14.1 AdminsDevicesController

Vedi §4.2.1.3.



4.2.14.2 InsertDeviceUseCase

Questa interfaccia modella il caso d'uso che si occupa dell'inserimento di una nuova macchina.

Metodi

- `void insertDevice(
 DeviceToInsert device);` la macchina da inserire.
 throws BusinessException
 Verifica che non esista già una macchina con lo stesso nome e, in caso affermativo, effettua la chiamata per l'inserimento della nuova macchina e, conseguentemente, quella per l'inserimento delle sue caratteristiche. Lancia un'eccezione se la macchina esiste già.

4.2.14.3 DeviceToInsert

Questo record rappresenta una macchina che deve essere memorizzata.

Campi privati

- `String username;` il nome della macchina;
- `List<DetailedCharacteristic> characteristics;` la lista delle caratteristiche della macchina.

4.2.14.4 NewCharacteristic

Questo record rappresenta i dati necessari per la creazione di una nuova caratteristica.

Campi privati

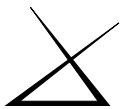
- `String name;` il nome della caratteristica;
- `OptionalDouble upperLimit;` il limite tecnico superiore. Può non essere specificato;
- `OptionalDouble lowerLimit;` il limite tecnico inferiore. Può non essere specificato;
- `boolean autoAdjust;` `true` se l'auto-adjust è attivo; `false` altrimenti;
- `OptionalInt sampleSize;` la grandezza del campione necessario a calcolare la media e la varianza. Può non essere specificato.

Metodi statici

- `NewCharacteristicBuilder builder()`
 Fornisce il builder del record con i seguenti valori di default:
 - `upperLimit: empty;`
 - `lowerLimit: empty;`
 - `sampleSize: empty.`

4.2.14.5 InsertDeviceService

Questa classe implementa l'inserimento di una nuova macchina.



Interfacce implementate

- `InsertDeviceUseCase`.

Campi privati

- `CreateDevice createDevice`: l'oggetto che si occupa di creare una nuova macchina;
- `InsertDevicePort insertDevicePort`: l'istanza di una porta a cui inoltrare le richieste per l'inserimento di una nuova macchina;
- `InsertCharacteristicUseCase insertCharacteristicUseCase`: l'istanza del caso d'uso che si occupa dell'inserimento di una nuova caratteristica.

Metodi pubblici

- `int insertDevice(
 DeviceToInsert device
) throws BusinessException`
Implementazione dell'omonimo metodo dell'interfaccia `InsertDeviceUseCase`.

4.2.14.6 CreateDevice

Questa classe si occupa della creazione di una nuova macchina.

Annotazioni

- `@Component`.

Metodi

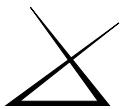
- `String generateApiKey()`
Restituisce una nuova chiave per l'API rilevazioni, randomica;
- `NewDevice createDevice(
 String name
)`
Restituisce una nuova macchina con il nome dato.

4.2.14.7 NewDevice

Questo record rappresenta una nuova macchina.

Campi privati

- `String name`: il nome della macchina;
- `boolean deactivate`: lo stato di attivazione della macchina;
- `boolean archived`: lo stato di archiviazione della macchina;
- `String apiKey`: la chiave della macchina per la API rilevazioni.



4.2.14.8 FindTinyDeviceByNamePort

Vedi §4.2.13.8.

4.2.14.9 TinyDevice

Vedi §4.2.5.3.

4.2.14.10 InsertDevicePort

Questa interfaccia rappresenta la porta per l'inserimento di una nuova macchina.

Metodi

- `int insertDevice(`
 `NewDevice device:` la macchina da memorizzare.
 `)`
 Memorizza la macchina.

4.2.14.11 AdminDeviceAdapter

Vedi §4.2.9.6.

4.2.14.12 DeviceRepository

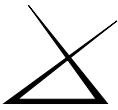
Vedi §4.2.5.7.

4.2.14.13 InsertCharacteristicUseCase

Questa interfaccia modella il caso d'uso che si occupa dell'inserimento di una nuova caratteristica in una macchina.

Metodi

- `int insertByDevice(`
 `int deviceId:` l'identificativo della macchina;
 `NewCharacteristic characteristic:` la nuova caratteristica da inserire.
 `) throws BusinessException`
Inserisce la nuova caratteristica nella macchina, controllando che questo sia possibile e che le informazioni date siano conformi ai requisiti, e ne restituisce l'identificativo.



4.2.15 Lista delle caratteristiche di una macchina

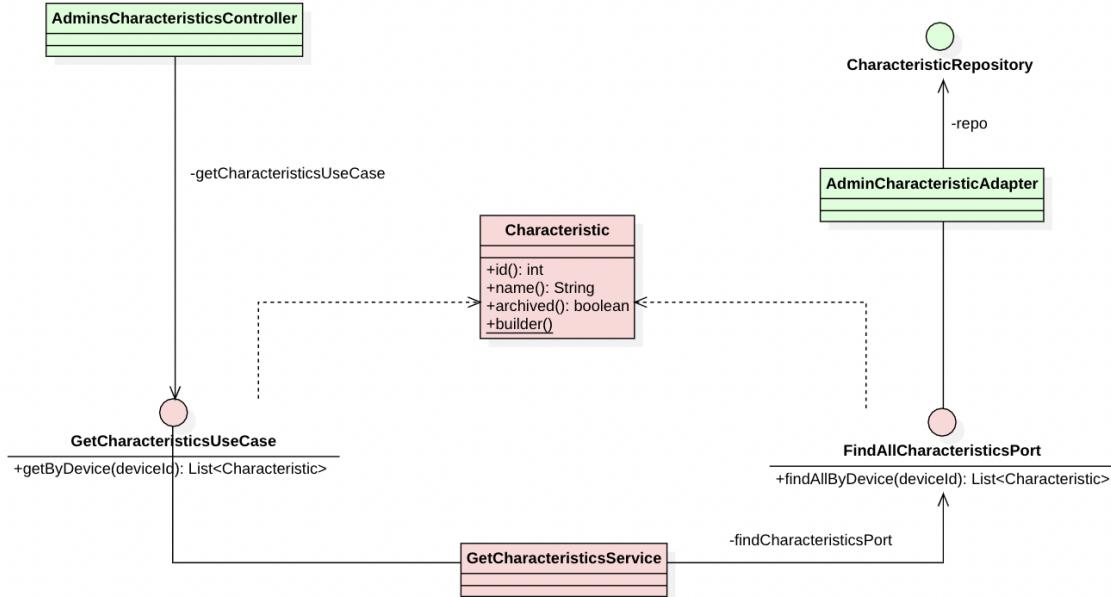


Figura 32: Diagramma delle classi relativo all'ottenimento della lista delle caratteristiche.

4.2.15.1 AdminsCharacteristicsController

Vedi §4.2.1.5.

4.2.15.2 GetCharacteristicsUseCase

Questa interfaccia modella il caso d'uso che si occupa dell'ottenimento della lista delle caratteristiche di una macchina.

Metodi

- `List<Characteristic> getByDevice(`
 `int deviceId: l'identificativo della macchina.`
 `) throws BusinessException`
 Restituisce la lista delle caratteristiche della macchina.

4.2.15.3 BusinessException

Vedi §4.2.2.2.

4.2.15.4 Characteristic

Questo record rappresenta l'intestazione di una caratteristica, con l'identificativo, il nome e il valore di `archived`.

Campi privati

- `int id`: l'identificativo della caratteristica, relativo alla macchina;
- `String name`: il nome della caratteristica;
- `boolean archived`: `true` se la caratteristica è archiviata; `false` altrimenti.

Metodi statici

- `CharacteristicBuilder builder()`
Fornisce il builder del record con `archived` inizializzato a `false` di default.

4.2.15.5 GetCharacteristicsService

Questa classe implementa l'ottenimento della lista delle caratteristiche di una macchina.

Interfacce implementate

- `GetCharacteristicsUseCase`.

Campi privati

- `FindAllCharacteristicsPort findCharacteristicsPort`: l'istanza della porta a cui inoltrare le richieste per l'ottenimento di tutte le caratteristiche di una macchina.

Metodi pubblici

- `List<Characteristic> getByDevice(`
`int deviceId`: l'identificativo della macchina.
)
throws `BusinessException`
Implementazione dell'omonimo metodo dell'interfaccia `GetCharacteristicsUseCase`.

4.2.15.6 FindAllCharacteristicsPort

Questa interfaccia modella la porta per l'ottenimento della lista delle caratteristiche di una macchina.

Metodi

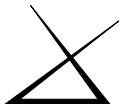
- `List<Characteristic> findAllById(`
`int deviceId`: l'identificativo della macchina.
)
Cerca le caratteristiche della macchina con l'identificativo dato e le restituisce.

4.2.15.7 AdminCharacteristicAdapter

Questa classe si occupa di adattare il repository di Spring alle porte descritte dalla parte di business per le operazioni relative alle caratteristiche da parte degli admin.

Annotazioni

- `@Component`.



Interfacce implementate

- `FindDetailedCharacteristicPort;`
- `InsertCharacteristicPort;`
- `FindAllCharacteristicsPort;`
- `UpdateCharacteristicPort.`

Campi privati

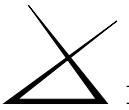
- `CharacteristicRepository repo:` un'istanza del repository delle caratteristiche.

Metodi pubblici

- `List<Characteristic> findAllByDeviceId(`
 `int deviceId:` l'identificativo della macchina.
 `)`
 Implementazione dell'omonimo metodo definito in `FindAllCharacteristicsPort`;
- `Optional<DetailedCharacteristic> findByCharacteristic(`
 `int deviceId:` l'identificativo della macchina;
 `int characteristicId:` l'identificativo della caratteristica, relativo alla macchina.
 `)`
 Implementazione dell'omonimo metodo definito in `FindDetailedCharacteristicPort`;
- `List<DetailedCharacteristic> findByDeviceAndName(`
 `int deviceId:` l'identificativo della macchina;
 `String name:` il nome della caratteristica.
 `)`
 Implementazione dell'omonimo metodo definito in `FindDetailedCharacteristicPort`;
- `int insertByDevice(`
 `int deviceId:` l'identificativo della macchina;
 `NewCharacteristic characteristic:` la caratteristica da inserire.
 `)`
 Implementazione dell'omonimo metodo definito in `InsertCharacteristicPort`;
- `void updateCharacteristic(`
 `DetailedCharacteristic characteristic:` la caratteristica da modificare.
 `)`
 Implementazione dell'omonimo metodo definito in `UpdateCharacteristicPort`.

4.2.15.8 CharacteristicRepository

Vedi §4.2.6.9.



4.2.16 Archiviazione caratteristica

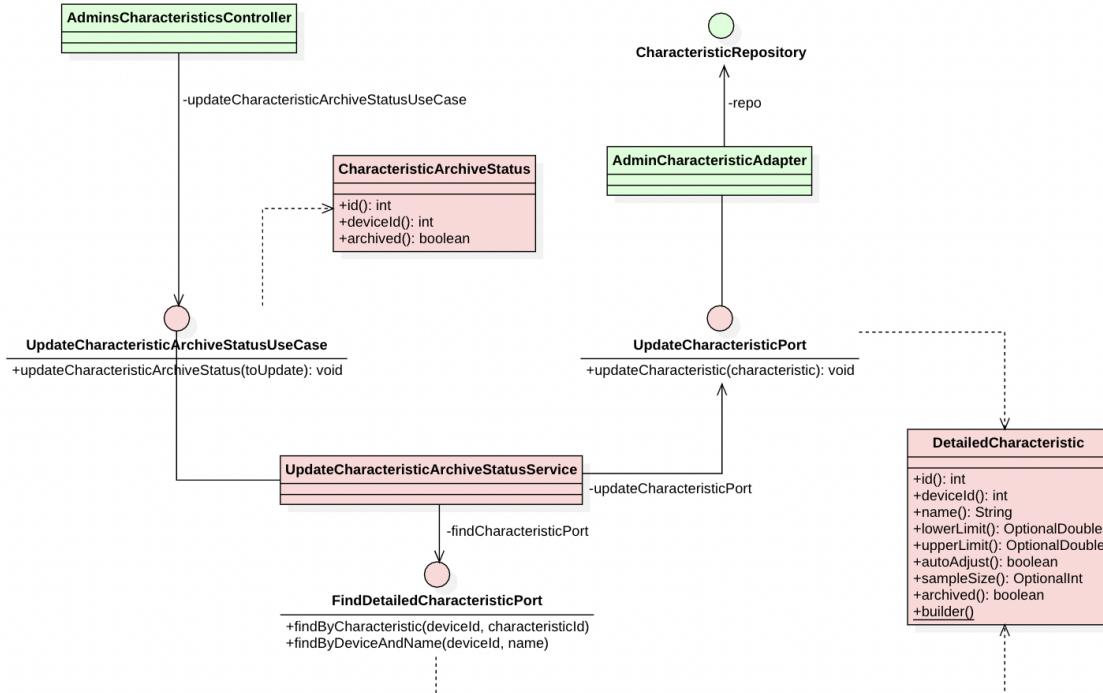


Figura 33: Diagramma delle classi relativo alla modifica dello stato di archiviazione di una caratteristica.

4.2.16.1 AdminsCharacteristicsController

Vedi §4.2.1.5.

4.2.16.2 UpdateCharacteristicArchiveStatusUseCase

Questa interfaccia modella il caso d'uso che si occupa della modifica dello stato di archiviazione di una caratteristica.

Metodi

- `void updateCharacteristicArchiveStatus(CharacteristicArchiveStatus toUpdate)`
 `CharacteristicArchiveStatus toUpdate`: la caratteristica da modificare.
 `) throws BusinessException`
 Verifica che la caratteristica esista e ne modifica lo stato di archiviazione con il valore dato.

4.2.16.3 CharacteristicArchiveStatus

Questo record rappresenta una caratteristica a cui è stato aggiornato lo stato di archiviazione.

Campi privati

- `int id`: l'identificativo della caratteristica, relativo alla macchina;
- `int deviceId`: l'identificativo della macchina;
- `boolean archived`: lo stato di archiviazione da impostare alla caratteristica.

4.2.16.4 UpdateCharacteristicArchiveStatusService

Questa classe implementa la modifica dello stato di archiviazione di una caratteristica.

Interfacce implementate

- `UpdateCharacteristicArchiveStatusUseCase`.

Campi privati

- `FindDetailedCharacteristicPort findCharacteristicPort`: l'istanza della porta a cui inoltrare le richieste per l'ottenimento di una caratteristica dato l'identificativo;
- `UpdateCharacteristicPort updateCharacteristicPort`: l'istanza della porta a cui inoltrare le richieste per la modifica di una caratteristica.

Metodi pubblici

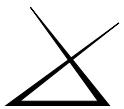
- `void updateCharacteristicArchiveStatus(`
 `CharacteristicArchiveStatus toUpdate`: la caratteristica da modificare.
 `) throws BusinessException`
 Implementazione dell'omonimo metodo dell'interfaccia
 `UpdateCharacteristicArchiveStatusUseCase`.

4.2.16.5 FindDetailedCharacteristicPort

Questa interfaccia modella la porta per la ricerca di una caratteristica, completa di tutte le sue informazioni.

Metodi

- `Optional<DetailedCharacteristic> findByCharacteristic(`
 `int deviceId`: l'identificativo della macchina;
 `int characteristicId`: l'identificativo della caratteristica, relativo alla macchina.
 `)`
 Restituisce tutti i dettagli della caratteristica di una macchina;
- `List<DetailedCharacteristic> findByDeviceAndName(`
 `int deviceId`: l'identificativo della macchina;
 `String name`: il nome della caratteristica.
 `)`
 Restituisce le caratteristiche di una macchina con un dato nome.



4.2.16.6 ConvertCharacteristic

Questa interfaccia fornisce i metodi per convertire un'istanza della classe `CharacteristicEntity` in una della classe `DetailedCharacteristic`, utilizzata nella logica di business, e viceversa.

Metodi statici

- `DetailedCharacteristic toDetailed(`

```
    CharacteristicEntity characteristic.
```

```
)
```

Converte un'istanza della classe `CharacteristicEntity` in una della classe `DetailedCharacteristic`;

- `CharacteristicEntity toEntity(`

```
    DetailedCharacteristic characteristic.
```

```
)
```

Converte un'istanza della classe `DetailedCharacteristic` in una della classe `CharacteristicEntity`.

4.2.16.7 UpdateCharacteristicPort

Questa interfaccia modella la porta per la modifica di una caratteristica.

Metodi

- `void updateCharacteristic(`

```
    DetailedCharacteristic characteristic: la caratteristica da modificare.
```

```
)
```

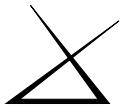
Modifica le informazioni di una caratteristica.

4.2.16.8 AdminCharacteristicAdapter

Vedi §4.2.15.7.

4.2.16.9 CharacteristicRepository

Vedi §4.2.6.9.



4.2.17 Inserimento caratteristica

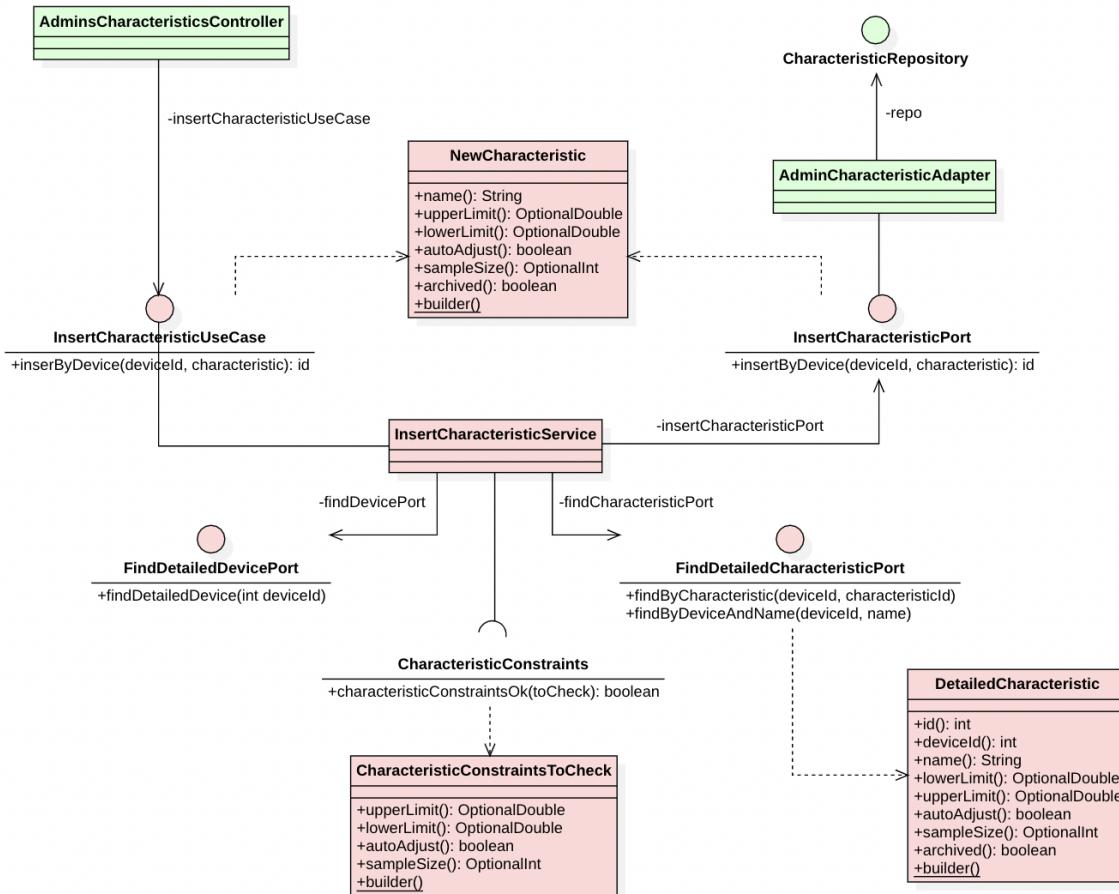


Figura 34: Diagramma delle classi relativo all'inserimento di una caratteristica.

4.2.17.1 AdminsCharacteristicsController

Vedi §4.2.1.5.

4.2.17.2 InsertCharacteristicUseCase

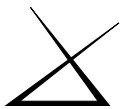
Vedi §4.2.14.13.

4.2.17.3 NewCharacteristic

Vedi §4.2.14.4.

4.2.17.4 InsertCharacteristicService

Questa classe implementa l'inserimento di una nuova caratteristica in una macchina.



Interfacce implementate

- `InsertCharacteristicUseCase`.

Campi privati

- `InsertCharacteristicPort insertCharacteristicPort`: l'istanza della porta a cui inoltrare le richieste per l'inserimento di una nuova caratteristica in una macchina;
- `FindDetailedDevicePort findDevicePort`: l'istanza della porta a cui inoltrare le richieste per l'ottenimento di una macchina;
- `FindDetailedCharacteristicPort findCharacteristicPort`: l'istanza della porta a cui inoltrare le richieste per l'ottenimento di una caratteristica.

Metodi pubblici

- `int insertByDevice(
 int deviceId: l'identificativo della macchina;
 NewCharacteristic characteristic: la nuova caratteristica da inserire.
) throws BusinessException`
Implementazione dell'omonimo metodo dell'interfaccia `InsertCharacteristicUseCase`.

4.2.17.5 CharacteristicConstraints

Vedi §4.2.18.6.

4.2.17.6 InsertCharacteristicPort

Questa interfaccia rappresenta la porta per l'inserimento di una nuova caratteristica in una macchina.

Metodi

- `int insertByDevice(
 int deviceId: l'identificativo della macchina;
 NewCharacteristic characteristic: la nuova caratteristica da inserire.
)`
Inserisce la nuova caratteristica nella macchina.

4.2.17.7 FindDetailedDevicePort

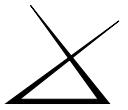
Vedi §4.2.10.5.

4.2.17.8 FindDetailedCharacteristicPort

Vedi §4.2.16.5.

4.2.17.9 ConvertCharacteristic

Vedi §4.2.16.6.



4.2.17.10 AdminCharacteristicAdapter

Vedi §4.2.15.7.

4.2.17.11 CharacteristicRepository

Vedi §4.2.6.9.

4.2.18 Modifica caratteristica

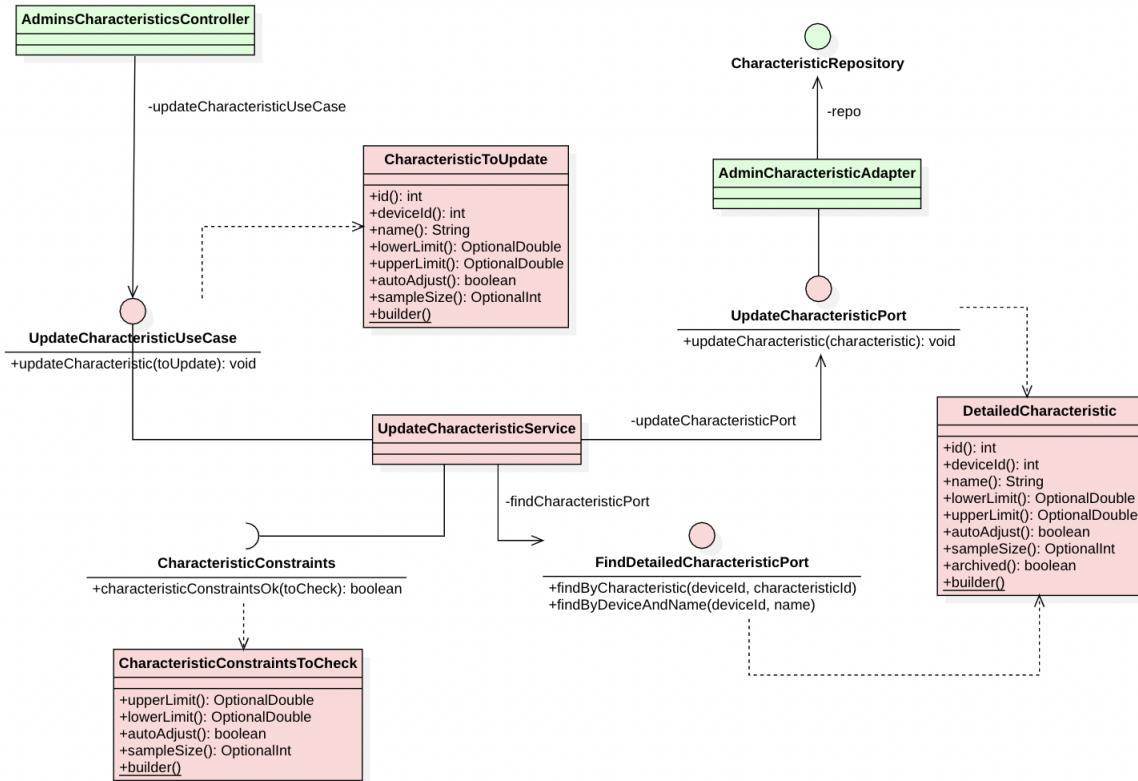


Figura 35: Diagramma delle classi relativo alla modifica di una caratteristica.

4.2.18.1 AdminsCharacteristicsController

Vedi §4.2.1.5.

4.2.18.2 UpdateCharacteristicUseCase

Questa interfaccia modella il caso d'uso che si occupa della modifica della caratteristica di una macchina.

Metodi

- `void updateCharacteristic()`

```
CharacteristicToUpdate toUpdate: la caratteristica da modificare.  
) throws BusinessException  
Modifica la caratteristica con le informazioni date, controllando che queste siano conformi ai  
requisiti.
```

4.2.18.3 CharacteristicToUpdate

Questo record rappresenta i dati necessari per la modifica di una caratteristica.

Campi privati

- `int id`: l'identificativo della caratteristica, relativo alla macchina;
- `int deviceId`: l'identificativo della macchina;
- `String name`: il nuovo nome della caratteristica;
- `OptionalDouble upperLimit`: il nuovo limite tecnico superiore. Può non essere specificato;
- `OptionalDouble lowerLimit`: il nuovo limite tecnico inferiore. Può non essere specificato;
- `boolean autoAdjust`: `true` se la caratteristica avrà l'auto-adjust attivo dopo la modifica; `false` altrimenti;
- `OptionalInt sampleSize`: la nuova grandezza del campione di rilevazioni necessario per calcolare la media, in caso di auto-adjust attivo. Può non essere specificato.

Metodi statici

- `CharacteristicToUpdateBuilder builder()`
Fornisce il builder del record con i valori dei campi `lowerLimit`, `upperLimit` e `sampleSize` non specificati.

4.2.18.4 BusinessException

Vedi §4.2.2.2.

4.2.18.5 UpdateCharacteristicService

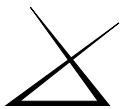
Questa classe implementa la modifica della caratteristica di una macchina.

Interfacce implementate

- `UpdateCharacteristicUseCase`.

Campi privati

- `FindDetailedCharacteristicPort findCharacteristicPort`: l'istanza della porta a cui inoltrare le richieste per l'ottenimento di una caratteristica;
- `UpdateCharacteristicPort updateCharacteristicPort`: l'istanza della porta a cui inoltrare le richieste per la modifica di una caratteristica.



Metodi pubblici

- void updateCharacteristic(

CharacteristicToUpdate toUpdate: la caratteristica da modificare.

-) throws BusinessException

Implementazione dell'omonimo metodo dell'interfaccia `UpdateCharacteristicUseCase`.

4.2.18.6 CharacteristicConstraints

Questa interfaccia fornisce il metodo che controlla che una caratteristica rispetti i requisiti relativi all'auto-adjust, i limiti tecnici e la grandezza del campione.

Metodi statici

- boolean characteristicConstraintsOk(

CharacteristicConstraintsToCheck toCheck: la caratteristica da controllare.

-)

Restituisce `true` se la caratteristica rispetta i requisiti relativi all'auto-adjust, i limiti tecnici e la grandezza del campione; `false`, altrimenti.

4.2.18.7 FindDetailedCharacteristicPort

Vedi §4.2.16.5.

4.2.18.8 ConvertCharacteristic

Vedi §4.2.16.6.

4.2.18.9 UpdateCharacteristicPort

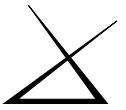
Vedi §4.2.16.7.

4.2.18.10 AdminCharacteristicAdapter

Vedi §4.2.15.7.

4.2.18.11 CharacteristicRepository

Vedi §4.2.6.9.



4.2.19 Lista utenti

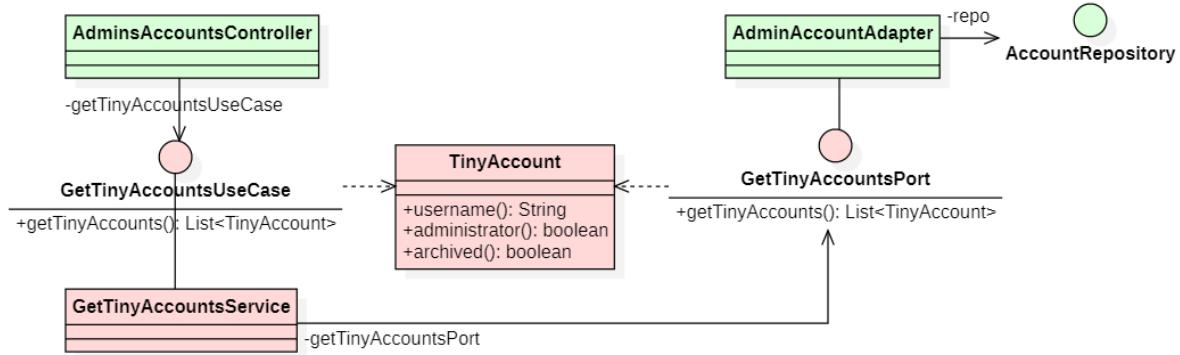


Figura 36: Diagramma delle classi relativo all'ottenimento della lista degli utenti.

4.2.19.1 AdminsAccountsController

Vedi §4.2.1.2.

4.2.19.2 GetTinyAccountsUseCase

Questa interfaccia modella il caso d'uso che si occupa dell'ottenimento della lista degli utenti con le informazioni essenziali.

Metodi

- `List<AccountTiny> getTinyAccounts()`
Restituisce la lista degli utenti.

4.2.19.3 TinyAccount

Questo record rappresenta un utente con tutte le sue informazioni, meno la password.

Campi privati

- `String username`: l'username dell'utente;
- `boolean administrator`: i permessi dell'utente;
- `boolean archived`: lo stato di archiviazione dell'utente.

4.2.19.4 GetTinyAccountsService

Questa classe implementa l'ottenimento della lista degli utenti con le informazioni essenziali.

Interfacce implementate

- `GetTinyAccountsUseCase`.

Campi privati

- `GetTinyAccountsPort getTinyAccountsPort`: l'istanza della porta a cui inoltrare le richieste per l'ottenimento della lista degli utenti.

Metodi pubblici

- `List<TinyAccount> getTinyAccounts()`.
Implementazione dell'omonimo metodo dell'interfaccia `GetTinyAccountsUseCase`.

4.2.19.5 GetTinyAccountsPort

Questa interfaccia modella la porta per l'ottenimento della lista degli utenti.

Metodi

- `List<TinyAccount> getTinyAccounts()`
Restituisce la lista degli utenti con le informazioni essenziali.

4.2.19.6 AdminAccountAdapter

Questa classe si occupa di adattare il repository di Spring alle porte descritte dalla parte di business per le operazioni effettuate dagli amministratori, riguardanti gli utenti.

Annotazioni

- `@Component`.

Interfacce implementate

- `UpdateAccountArchiveStatusPort`;
- `UpdateAccountByAdminPort`;
- `FindAccountByAdminPort`;
- `InsertAccountPort`;
- `GetTinyAccountsPort`.

Campi privati

- `AccountRepository repo`: un'istanza del repository degli utenti.

Metodi pubblici

- `void updateAccountArchiveStatus(`

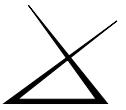
`Account account`: l'utente con lo stato di archiviazione aggiornato.

)

Memorizza l'utente con lo stato di archiviazione aggiornato;

- `Optional<Account> findByUsername(`

`String username`: l'username dell'utente da cercare.



```
)  
Cerca l'utente e lo restituisce se lo trova;  
• void updateAccount(  
    Account account: l'utente da cercare.  
)  
Memorizza l'utente aggiornato;  
• void insertAccount(  
    Account account: l'utente da memorizzare.  
)  
Memorizza il nuovo utente;  
• List<TinyAccount> getTinyAccounts()  
    Restituisce gli utenti memorizzati.
```

4.2.19.7 AccountRepository

Vedi §4.2.3.10.

4.2.20 Archiviazione utente

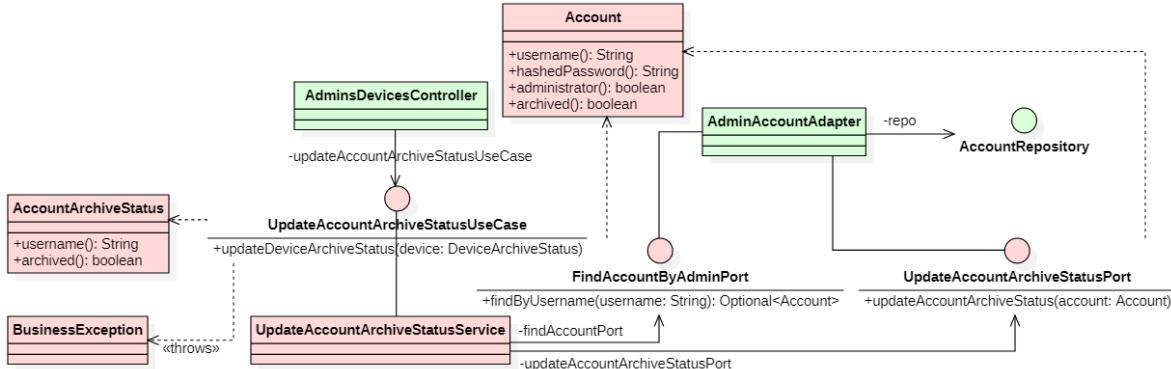


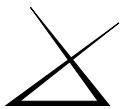
Figura 37: Diagramma delle classi relativo alla modifica dello stato di archiviazione di un utente.

4.2.20.1 AdminsAccountsController

Vedi §4.2.1.2.

4.2.20.2 UpdateAccountArchiveStatusUseCase

Questa interfaccia modella il caso d'uso che si occupa della modifica dello stato di archiviazione di un utente.



Metodi

- `void updateAccountArchiveStatus(`

`AccountArchiveStatus account`: l'utente con lo stato di archiviazione modificato.

- `) throws BusinessException`

Verifica che l'utente esista e modifica lo stato di archiviazione con il valore assegnato. Lancia un'eccezione se l'utente non esiste.

4.2.20.3 AccountArchiveStatus

Questo record rappresenta un utente con stato di archiviazione modificato da un amministratore.

Campi privati

- `String username`: l'username dell'utente da modificare;
- `boolean archived`: lo stato di archiviazione dell'utente da modificare.

4.2.20.4 BusinessException

Vedi §4.2.2.2.

4.2.20.5 UpdateAccountArchiveStatusService

Questa classe implementa la modifica dello stato di archiviazione di un utente.

Interfacce implementate

- `UpdateAccountArchiveStatusUseCase`.

Campi privati

- `UpdateAccountArchiveStatusPort updateAccountArchiveStatusPort`: l'istanza della porta a cui inoltrare le richieste per l'aggiornamento dello stato di archiviazione di un utente;
- `FindAccountByAdminPort findAccountByAdminPort`: l'istanza della porta a cui inoltrare le richieste per la ricerca di un utente dato l'username, per conto di un amministratore.

Metodi pubblici

- `void updateAccountArchiveStatus(`

`AccountArchiveStatus account`: l'utente con lo stato di archiviazione modificato.

- `) throws BusinessException`

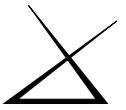
Implementazione dell'omonimo metodo dell'interfaccia `UpdateAccountArchiveStatusUseCase`.

4.2.20.6 Account

Vedi §4.2.3.2.

4.2.20.7 FindAccountByAdminPort

Questa interfaccia rappresenta la porta per la ricerca di un utente all'interno dello strato di persistenza.



Metodi

- `Optional<Account> findByUsername(`

`String username`: l'username dell'utente da cercare.

)

Cerca l'utente e, se lo trova, lo restituisce; altrimenti restituisce `Optional` vuoto.

4.2.20.8 UpdateAccountArchiveStatusPort

Questa interfaccia modella la porta per la modifica dello stato di archiviazione di un utente.

Metodi

- `void updateAccountArchiveStatus(`

`Account account`: l'utente con lo stato di archiviazione modificato.

)

Memorizza il nuovo stato di archiviazione dell'utente.

4.2.20.9 AdminAccountAdapter

Vedi §4.2.19.6.

4.2.20.10 AccountRepository

Vedi §4.2.3.10.

4.2.21 Inserimento utente

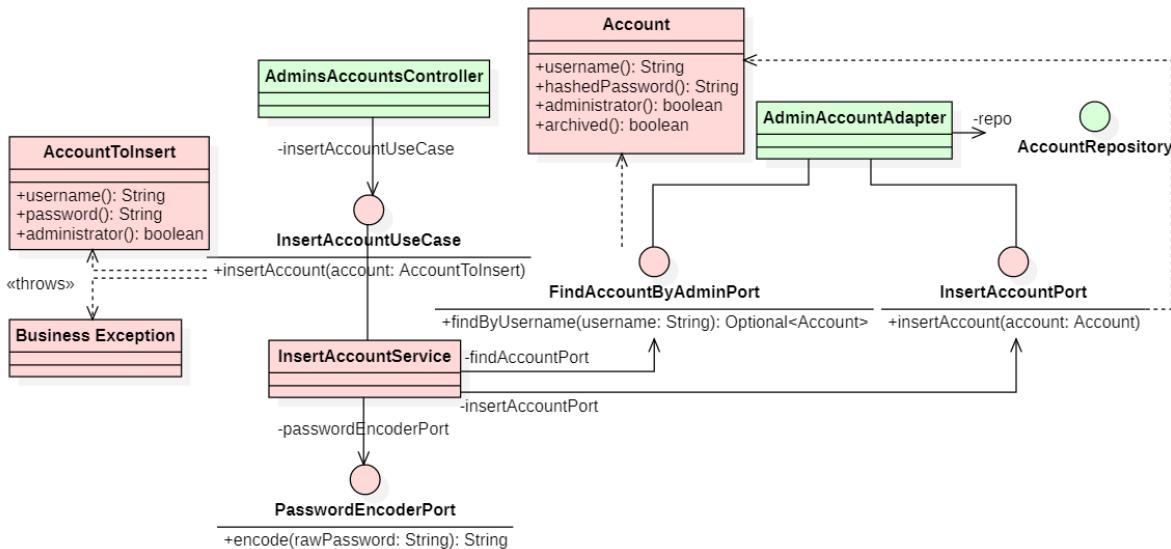
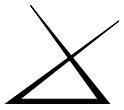


Figura 38: Diagramma delle classi relative all'inserimento di un utente.



4.2.21.1 AdminsAccountsController

Vedi §4.2.1.2.

4.2.21.2 InsertAccountUseCase

Questa interfaccia modella il caso d'uso che si occupa dell'inserimento di un nuovo utente.

Metodi

- `void insertAccount(`

`AccountToInsert accountToInsert:` l'utente da inserire.

- `) throws BusinessException`

Verifica che la nuova password, se presente, soddisfi i requisiti di ammissibilità, che non esista già un utente con lo stesso username e, in caso affermativo effettua la chiamata per l'inserimento del nuovo utente. Lancia un'eccezione se la password non è valida o l'utente esiste già.

4.2.21.3 AccountToInsert

Questo record rappresenta un utente da inserire.

Campi privati

- `String username:` l'username dell'utente da inserire;
- `String password:` la password dell'utente da inserire;
- `boolean administrator:` i permessi dell'utente da inserire.

4.2.21.4 BusinessException

Vedi §4.2.2.2.

4.2.21.5 InsertAccountService

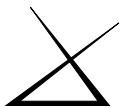
Questa classe implementa l'inserimento di un nuovo utente.

Interfacce implementate

- `InsertAccountUseCase.`

Campi privati

- `InsertAccountPort insertAccountPort:` l'istanza della porta a cui inoltrare le richieste per l'inserimento di un utente;
- `FindAccountByAdminPort findAccountByAdminPort:` l'istanza della porta a cui inoltrare le richieste per la ricerca di un utente dato l'username, per conto di un amministratore;
- `PasswordEncoderPort passwordEncoderPort:` l'istanza della porta a cui inoltrare le richieste per la cifratura di una password in chiaro.



Metodi pubblici

- void insertAccount(

AccountToInsert accountToInsert: l'utente da inserire.

-) throws BusinessException

Implementazione dell'omonimo metodo dell'interfaccia `InsertAccountUseCase`.

4.2.21.6 FindAccountByAdminPort

Vedi §4.2.20.7.

4.2.21.7 Account

Vedi §4.2.3.2.

4.2.21.8 InsertAccountPort

Questa interfaccia rappresenta la porta che si occupa dell'inserimento di un nuovo utente.

Metodi

- void insertAccount(

Account account: l'utente da inserire.

-)

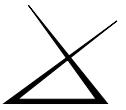
Memorizza il nuovo utente.

4.2.21.9 AdminAccountAdapter

Vedi §4.2.19.6.

4.2.21.10 AccountRepository

Vedi §4.2.3.10.



4.2.22 Modifica utente

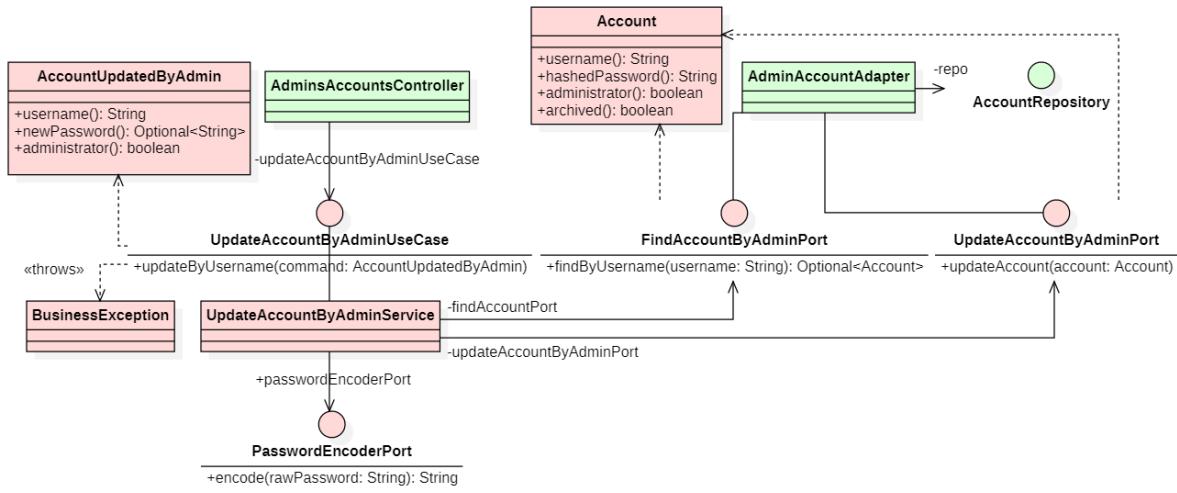


Figura 39: Diagramma delle classi relative alla modifica dell’utente per mano di un amministratore.

4.2.22.1 AdminsAccountsController

Vedi §4.2.1.2.

4.2.22.2 UpdateAccountByAdminUseCase

Questa interfaccia modella il caso d’uso che si occupa della modifica di un utente per mano di un amministratore.

Metodi

- void updateByUsername(
 AccountUpdatedByAdmin updatedAccount: l’utente modificato.
) throws BusinessException
 Verifica che la nuova password, se presente, soddisfi i requisiti di ammissibilità, che l’utente esista e in caso affermativo di effettuare le chiamate per l’aggiornamento dei suoi dati. Lancia un’eccezione se la password non è valida o l’utente non è stato trovato.

4.2.22.3 AccountUpdatedByAdmin

Questo record rappresenta un utente modificato da un amministratore.

Campi privati

- String username: l’username dell’utente da modificare;
- Optional<String> password: optionalmente, la nuova password da impostare all’utente;
- boolean administrator: i permessi da impostare all’utente.

4.2.22.4 BusinessException

Vedi §4.2.2.2.

4.2.22.5 UpdateAccountByAdminService

Questa classe implementa la modifica di un utente per mano di un amministratore.

Interfacce implementate

- `UpdateAccountByAdminUseCase`.

Campi privati

- `UpdateAccountByAdminPort updateAccountByAdminPort`
L'istanza di una porta a cui inoltrare le richieste per l'aggiornamento dei dati di un utente;
- `FindAccountByAdminPort findAccountByAdminPort`
L'istanza di una porta a cui inoltrare le richieste per la ricerca di un utente dato l'username;
- `PasswordEncoderPort passwordEncoderPort`
L'istanza di una porta a cui inoltrare le richieste per la cifratura di una password in chiaro.

Metodi pubblici

- `void updateByUsername(`
`AccountUpdatedByAdmin updatedAccount`: l'utente con le modifiche da apportare.
`) throws BusinessException`
Implementazione dell'omonimo metodo dell'interfaccia `UpdateAccountByAdminUseCase`.

4.2.22.6 PasswordEncoderPort

Vedi §4.2.4.8.

4.2.22.7 FindAccountByAdminPort

Vedi §4.2.20.7.

4.2.22.8 Account

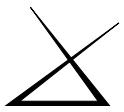
Vedi §4.2.3.2.

4.2.22.9 UpdateAccountByAdminPort

Questa interfaccia rappresenta la porta per l'aggiornamento dei dati di un utente per mano di un amministratore.

Metodi

- `void updateAccount(`
`Account account`: l'utente con le informazioni aggiornate.
`)`
Memorizza le modifiche fatte all'utente.



4.2.22.10 AdminAccountAdapter

Vedi §4.2.19.6.

4.2.22.11 AccountRepository

Vedi §4.2.3.10.

4.2.23 Configuration

Questo sottosezione è dedicata alla descrizione delle classi che si occupano di gestire la dependency injection delle classi di business, le quali non potendo dipendere dal framework Spring Boot non possono godere della dependency injection automatica.

4.2.23.1 AccountsConfiguration

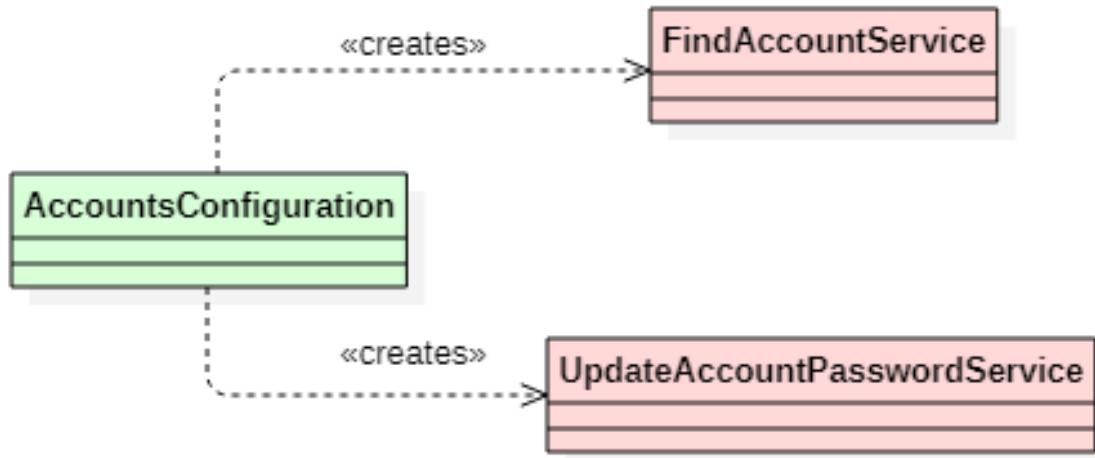


Figura 40: Diagramma delle classi relative a `AccountsConfiguration`.

Questa classe implementa una configurazione di Spring che descrive come creare le varie classi di business relative agli utenti.

Annotazioni

- `@Component`.

Metodi

- `@Bean`

```
FindAccountUseCase findAccountUseCase(
```

```
    FindAccountPort findAccountPort: la FindAccountPort da usare nel costruttore di
    FindAccountService.
```

```

)
Questo metodo crea un'istanza di FindAccountUseCase;

• @Bean
UpdateAccountPasswordUseCase updateAccountPasswordUseCase(
    FindAccountPort findAccountPort: la FindAccountPort da usare nel costruttore di
    UpdateAccountPasswordService;
    PasswordMatcherPort passwordMatcherPort: la PasswordMatcherPort da usare nel co-
    struttore di UpdateAccountPasswordService;
    PasswordEncoderPort passwordEncoderPort: la PasswordEncoderPort da usare nel co-
    struttore di UpdateAccountPasswordService;
    UpdateAccountPasswordPort updateAccountPasswordPort: la
    UpdateAccountPasswordPort da usare nel costruttore di UpdateAccountPasswordService.

)
Questo metodo crea un'istanza di UpdateAccountPasswordUseCase.

```

4.2.23.2 AdminsAccountsConfiguration

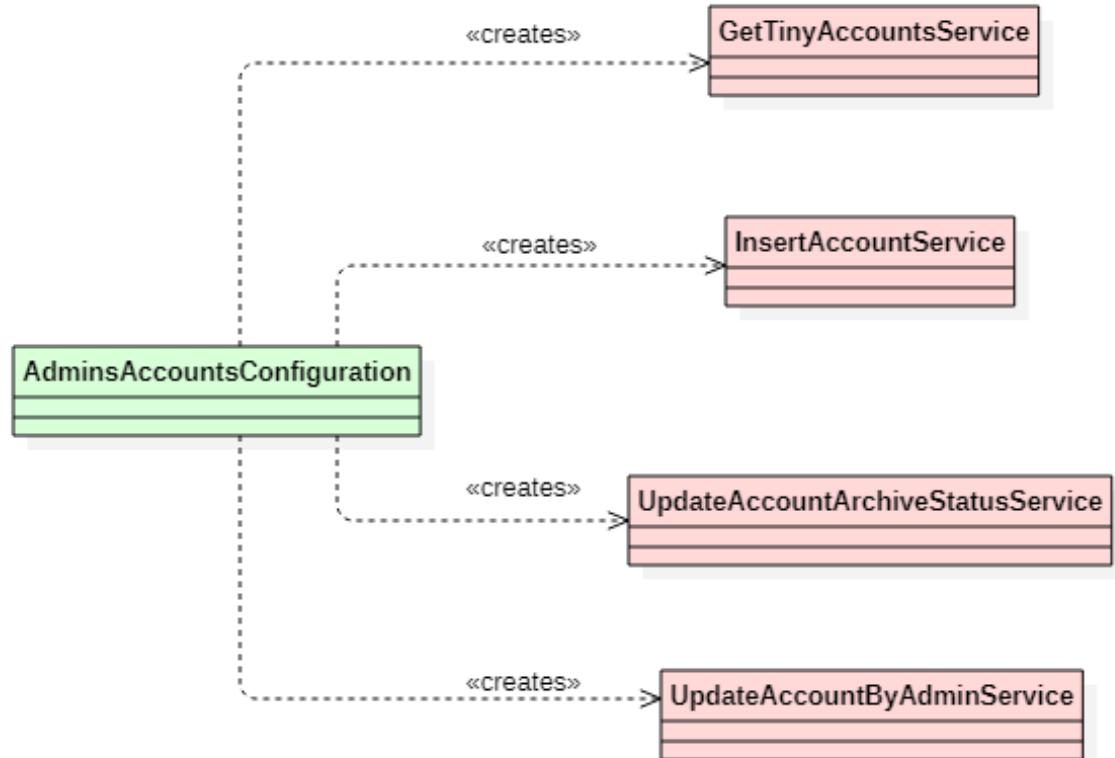


Figura 41: Diagramma delle classi relative a **AdminsAccountsConfiguration**.

Questa classe implementa una configurazione di Spring che descrive come creare le varie classi di business relative alle operazioni svolte dagli amministratori sugli utenti.

Annotazioni

- @Component.

Metodi

- @Bean

```
GetTinyAccountsUseCase getTinyAccountsUseCase(
```

GetTinyAccountsPort getAccountsPort: la GetTinyAccountsPort da usare nel costruttore di GetTinyAccountsService.

```
)
```

Questo metodo crea un'istanza di GetTinyAccountsUseCase;

- @Bean

```
InsertAccountUseCase insertAccountUseCase(
```

FindAccountByAdminPort findAccountByAdminPort: la FindAccountByAdminPort da usare nel costruttore di InsertAccountService;

PasswordEncoderPort passwordEncoderPort: la PasswordEncoderPort da usare nel costruttore di InsertAccountService;

InsertAccountPort insertAccountPort: la InsertAccountPort da usare nel costruttore di InsertAccountService.

```
)
```

Questo metodo crea un'istanza di InsertAccountUseCase;

- @Bean

```
UpdateAccountArchiveStatusUseCase updateAccountArchiveStatusUseCase(
```

FindAccountByAdminPort findAccountByAdminPort: la FindAccountByAdminPort da usare nel costruttore di UpdateAccountArchiveStatusService;

UpdateAccountArchiveStatusPort updateAccountArchiveStatusPort: la UpdateAccountArchiveStatusPort da usare nel costruttore di UpdateAccountArchiveStatusService.

```
)
```

Questo metodo crea un'istanza di UpdateAccountArchiveStatusUseCase;

- @Bean

```
UpdateAccountByAdminUseCase updateAccountByAdminUseCase(
```

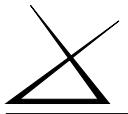
FindAccountByAdminPort findAccountByAdminPort: la FindAccountByAdminPort da usare nel costruttore di UpdateAccountByAdminService;

PasswordEncoderPort passwordEncoderPort: la PasswordEncoderPort da usare nel costruttore di UpdateAccountByAdminService;

UpdateAccountByAdminPort updateAccountByAdminPort: la UpdateAccountByAdminPort da usare nel costruttore di UpdateAccountByAdminService.

```
)
```

Questo metodo crea un'istanza di UpdateAccountByAdminUseCase.



4.2.23.3 AdminsDevicesConfiguration

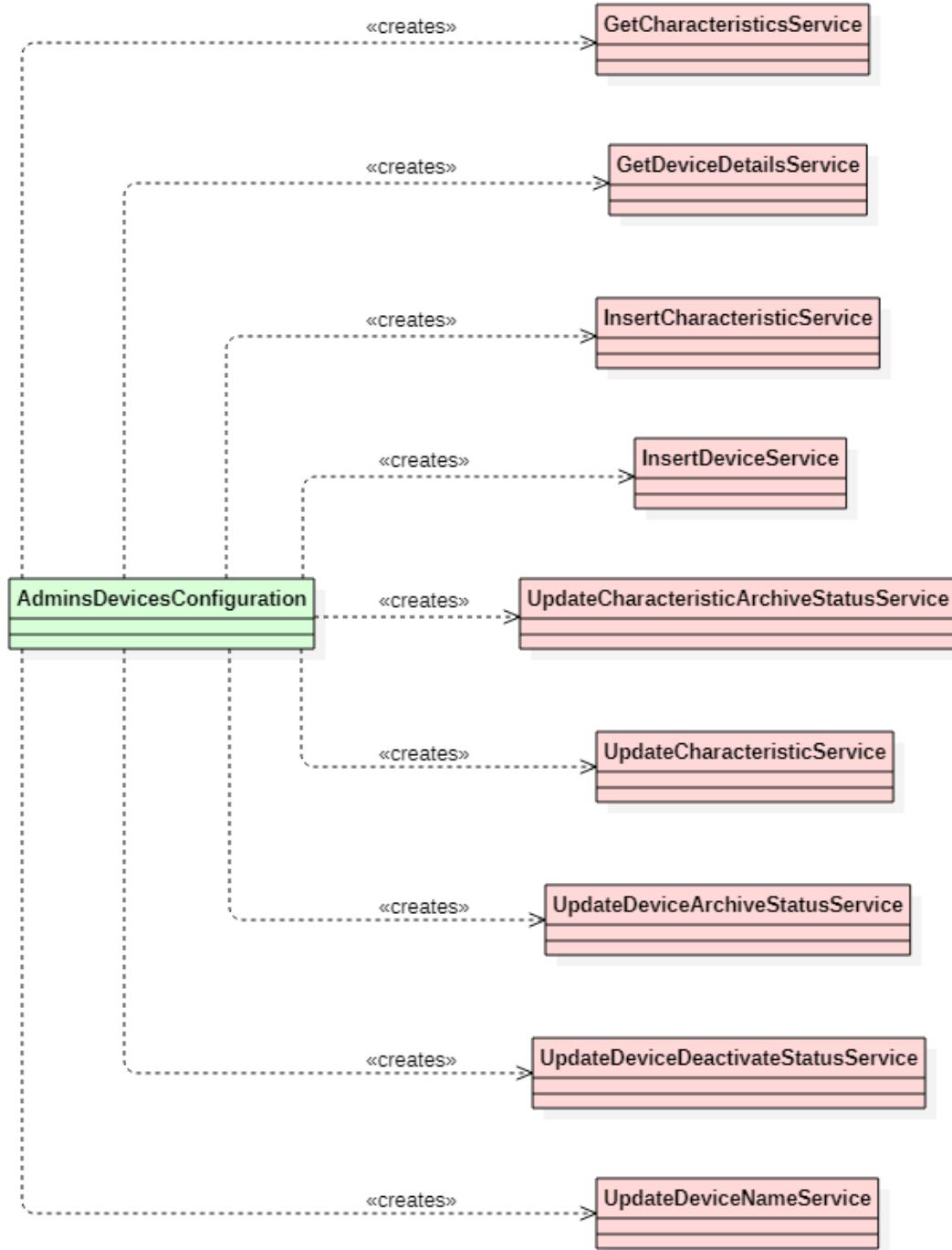
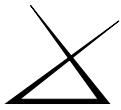


Figura 42: Diagramma delle classi relative a **AdminsDevicesConfiguration**.



Questa classe implementa una configurazione di Spring che descrive come creare le varie classi di business relative alle operazioni svolte dagli amministratori sulle macchine.

Annotazioni

- `@Component`.

Metodi

- `@Bean`

```
GetCharacteristicsUseCase getCharacteristicsUseCase(  
    FindAllCharacteristicsPort findAllCharacteristicsPort: la  
    FindAllCharacteristicsPort da usare nel costruttore di GetCharacteristicsService.  
)  
Questo metodo crea un'istanza di GetCharacteristicsUseCase;
```

- `@Bean`

```
GetDeviceDetailsUseCase getDeviceDetailsUseCase(  
    FindDetailedDevicePort findDetailedDevicePort): la FindDetailedDevicePort da  
    usare nel costruttore di  
    GetDeviceDetailsService.  
)  
Questo metodo crea un'istanza di GetDeviceDetailsUseCase;
```

- `@Bean`

```
GetDevicesUseCase getDevicesUseCase(  
    GetDevicesPort getDevicesPort: la GetDevicesPort da usare nel costruttore di  
    GetDevicesService.  
)
```

Questo metodo crea un'istanza di GetDevicesUseCase;

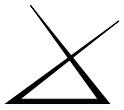
- `@Bean`

```
InsertCharacteristicUseCase insertCharacteristicUseCase(  
    InsertCharacteristicPort insertCharacteristicPort: la InsertCharacteristicPort  
    da usare nel costruttore di InsertCharacteristicService;  
    FindDetailedDevicePort findDevicePort: la FindDetailedDevicePort da usare nel  
    costruttore di InsertCharacteristicService;  
    FindDetailedCharacteristicPort findCharacteristicPort: la  
    FindDetailedCharacteristicPort da usare nel costruttore di  
    InsertCharacteristicService.  
)
```

Questo metodo crea un'istanza di InsertCharacteristicUseCase;

- `@Bean`

```
InsertDeviceUseCase insertDeviceUseCase(  
    FindTinyDeviceByNamePort findTinyDeviceByNamePort: la FindTinyDeviceByNamePort  
    da usare nel costruttore di InsertDeviceService;
```



```
InsertDevicePort insertDevicePort: la InsertDevicePort da usare nel costruttore di
InsertCharacteristicService;

CreateDevice createDevice: l'istanza di CreateDevice da usare nel costruttore di
InsertDeviceService;

InsertCharacteristicUseCase insertCharacteristicUseCase: lo
InsertCharacteristicUseCase da usare nel costruttore di InsertDeviceService.

)

Questo metodo crea un'istanza di InsertDeviceUseCase;

• @Bean
UpdateCharacteristicArchiveStatusUseCase updateCharacteristicArchiveStatusUseCase(

    FindDetailedCharacteristicPort findCharacteristicPort: la
    FindDetailedCharacteristicPort da usare nel costruttore di
    UpdateCharacteristicArchiveStatusService;

    UpdateCharacteristicPort updateCharacteristicPort: la UpdateCharacteristicPort
    da usare nel costruttore di UpdateCharacteristicArchiveStatusService.

)

Questo metodo crea un'istanza di UpdateCharacteristicArchiveStatusUseCase;

• @Bean
UpdateCharacteristicUseCase updateCharacteristicUseCase(

    FindDetailedCharacteristicPort findCharacteristicPort: la
    FindDetailedCharacteristicPort da usare nel costruttore di UpdateCharacteristicService;

    UpdateCharacteristicPort updateCharacteristicPort: la UpdateCharacteristicPort
    da usare nel costruttore di UpdateCharacteristicService.

)

Questo metodo crea un'istanza di UpdateCharacteristicUseCase;

• @Bean
UpdateDeviceArchiveStatusUseCase updateDeviceArchiveStatusUseCase(

    FindDetailedDevicePort findDetailedDevicePort: la
    FindDetailedDevicePort da usare nel costruttore di UpdateDeviceArchiveStatusService;

    UpdateDeviceArchiveStatusPort updateDeviceArchiveStatus: la
    UpdateDeviceArchiveStatusPort da usare nel costruttore di
    UpdateDeviceArchiveStatusService.

)

Questo metodo crea un'istanza di UpdateDeviceArchiveStatusUseCase;

• @Bean
UpdateDeviceDeactivateStatusUseCase updateDeviceDeactivateStatusUseCase(

    FindDetailedDevicePort findDetailedDevicePort: la FindDetailedDevicePort da usa-
    re nel costruttore di UpdateDeviceDeactivateStatusService;

    UpdateDeviceDeactivateStatusPort updateDeviceDeactivateStatusPort: la
    UpdateDeviceDeactivateStatusPort da usare nel costruttore di
    UpdateDeviceDeactivateStatusService.
```

```

        )
        Questo metodo crea un'istanza di UpdateDeviceDeactivateStatusUseCase;

    • @Bean
        UpdateDeviceNameUseCase updateDeviceNameUseCase(
            FindDetailedDevicePort findDetailedDevicePort: la FindDetailedDevicePort da usare nel costruttore di UpdateDeviceNameService;
            FindTinyDeviceByNamePort findTinyDeviceByNamePort: la FindTinyDeviceByNamePort da usare nel costruttore di UpdateDeviceNameService;
            UpdateDeviceNamePort updateDeviceNamePort: la UpdateDeviceNamePort da usare nel costruttore di UpdateDeviceNameService.
        )
        Questo metodo crea un'istanza di UpdateDeviceNameUseCase.
    
```

4.2.23.4 DetectionsConfiguration

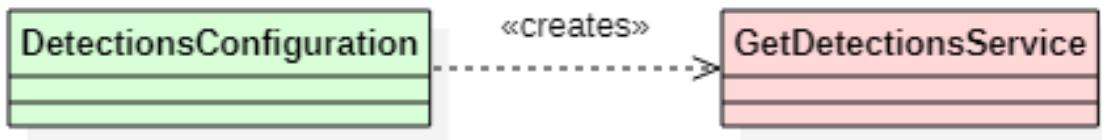


Figura 43: Diagramma delle classi relative a `DetectionsConfiguration`.

Questa classe implementa una configurazione di Spring che descrive come creare le varie classi di business relative alle rilevazioni.

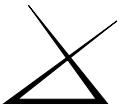
Annotazioni

- `@Component`.

Metodi

```

    • @Bean
        GetDetectionsUseCase getDetectionsUseCase(
            FindAllDetectionsPort findDetectionsPort: la FindAllDetectionsPort da usare nel costruttore di GetDetectionsService;
            FindCharacteristicLimitsPort findCharacteristicPort: la FindCharacteristicLimitsPort da usare nel costruttore di GetDetectionsService.
        )
        Questo metodo crea un'istanza di GetDetectionsUseCase.
    
```



4.2.23.5 DevicesConfiguration

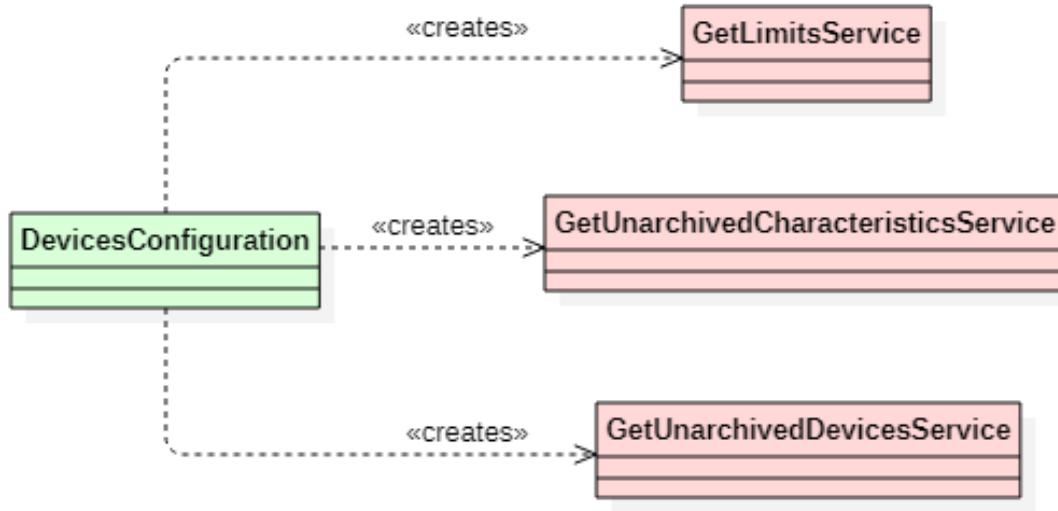


Figura 44: Diagramma delle classi relative a **DevicesConfiguration**.

Questa classe implementa una configurazione di Spring che descrive come creare le varie classi di business relative alle macchine.

Annotazioni

- `@Component`.

Metodi

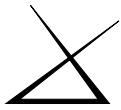
- `@Bean`

```
GetLimitsUseCase getLimitsUseCase(
    GetUnarchivedDevicesPort findDevicesPort: la GetUnarchivedDevicesPort da usare
    nel costruttore di GetLimitsService;
    FindCharacteristicLimitsPort findCharacteristicLimitsPort: la
    FindCharacteristicLimitsPort da usare nel costruttore di GetLimitsService.
)
```

Questo metodo crea un'istanza di `GetLimitsUseCase`;

- `@Bean`

```
GetUnarchivedCharacteristicsUseCase getUnarchivedCharacteristicsUseCase(
    GetUnarchivedDevicesPort findDevicesPort: la GetUnarchivedDevicesPort da usare
    nel costruttore di GetUnarchivedCharacteristicsService;
    FindAllUnarchivedCharacteristicsPort findCharacteristicsPort: la
    FindAllUnarchivedCharacteristicsPort da usare nel costruttore di
    GetUnarchivedCharacteristicsService.
)
```



)

Questo metodo crea un'istanza di `GetUnarchivedCharacteristicsUseCase`;

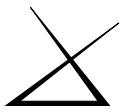
- `@Bean`

```
GetUnarchivedDevicesUseCase getUnarchivedDevices(
```

`GetUnarchivedDevicesPort getUnarchivedDevicesPort: la GetUnarchivedDevicesPort
da usare nel costruttore di
GetUnarchivedDevicesService.`

)

Questo metodo crea un'istanza di `GetUnarchivedDevicesUseCase`.



4.3 API rilevazioni

Come la componente back-end UI, anche la componente API rilevazioni è sviluppata in Java utilizzando il framework Spring Boot versione 2.6.2. La progettazione segue il pattern dell'architettura esagonale. I file sorgenti sono raggruppati nella cartella `src` e organizzati secondo la seguente struttura:

- `main/java`: contiene i sorgenti del software che sarà eseguito;
- `main/resources`: contiene la configurazione del software che andrà eseguito;
- `test/java`: contiene i sorgenti dei test;
- `test/resources`: contiene la configurazione del software da usare durante i test.

In particolare la cartella `main/java` contiene il package `it.deltax.produlytics.api`, il quale contiene a sua volta i seguenti package:

- `repositories`: contiene le interfacce relative alla comunicazione con il database;
- `exceptions`: contiene le classi relative alla gestione delle eccezioni e degli errori;
- `detections`: contiene le classi e interfacce relative all'endpoint `/detections` offerto. Esso contiene a sua volta i seguenti package:
 - `web`: contiene le classi relative ai controller, ovvero quelle con il compito di ricevere le richieste fatte all'endpoint;
 - `adapters`: contiene le classi il cui compito è adattare i repository alle interfacce di dominio;
 - `business`: contiene le classi e interfacce di business, indipendenti dal framework usato, e a sua volta contiene i seguenti package:
 - * `ports/in`: contiene le interfacce dei casi d'uso forniti dalla logica di business;
 - * `ports/out`: contiene le interfacce delle porte richieste dalla logica di business;
 - * `services`: contiene l'implementazione dei servizi di business;
 - * `domain`: contiene tutte le classi e interfacce utilizzate all'interno dei servizi di business.

La cartella `test/java` contiene il package `it.deltax.produlytics.api`, il quale contiene poi due package, `unit` e `integration`, che dividono i test di unità da quelli di integrazione.

Per ogni classe in questa sezione sono impliciti:

- l'esistenza di un costruttore che accetta un parametro per campo e lo assegna al relativo campo, a meno che non siano definiti altri costruttori;
- l'esistenza di un getter per ogni campo di un record, con lo stesso nome del relativo campo.

4.3.1 Controller

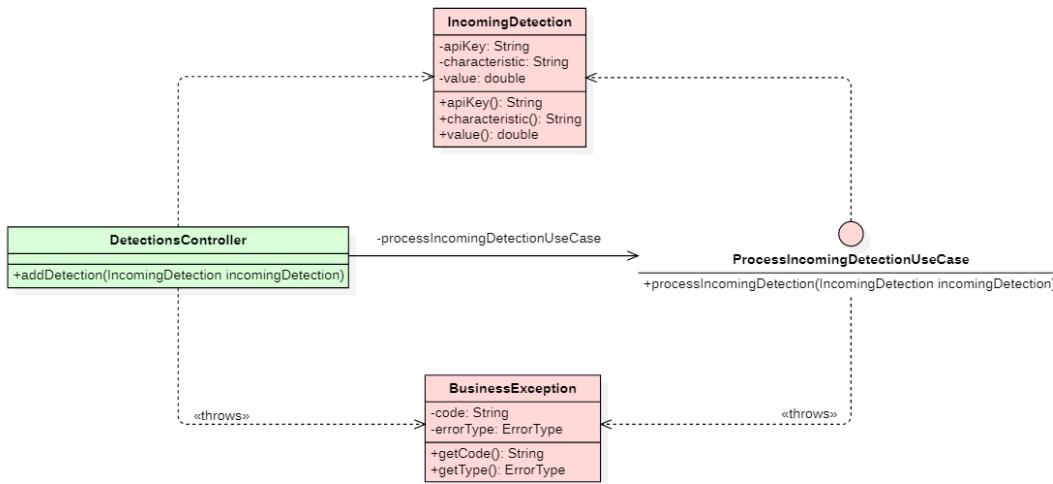


Figura 45: Diagramma delle classi relative a `DetectionsController`.

Questo modulo si occupa di ricevere le richieste fatte all'endpoint `/detections` e inoltrarle al servizio di business per essere processate. Il ruolo principale è svolto da un `RestController` di Spring Boot che riceve automaticamente le richieste fatte all'endpoint e già decodificate.

4.3.1.1 DetectionsController

Questa classe è un controller di Spring Boot. Il suo ruolo è essere il punto d'entrata delle richieste HTTP all'endpoint descritto in §3.4.3.1.1 e inoltrarle al service della parte di business.

Annotazioni

- `@RestController`;
- `@RequestMapping("/detections")`.

Campi privati

- `ProcessIncomingDetectionUseCase processIncomingDetectionUseCase`: l'istanza di un servizio a cui inoltrare le nuove rilevazioni in entrata e che si occuperà di processarle.

Metodi

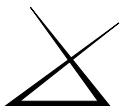
```

    • @PostMapping("")
    @ResponseStatus(HttpStatus.ACCEPTED)
    void addDetection(
    
```

`@RequestBody IncomingDetection incomingDetection`: il body della richiesta HTTP.

`) throws BusinessException`

Questo metodo riceve le richieste HTTP fatte all'API e si occupa di inoltrarle al relativo servizio di business.



4.3.1.2 ProcessIncomingDetectionUseCase

Questa interfaccia modella il caso d'uso che si occupa di gestire le rilevazioni provenienti da una macchina.

Metodi

- `void processIncomingDetection()`

`IncomingDetection incomingDetection`: la rilevazione in arrivo da processare.

- `) throws BusinessException`

Questo metodo gestisce le nuove rilevazioni, in particolare le autentica, le persiste e aggiorna quelle anomale secondo le carte di controllo. La persistenza e l'aggiornamento delle anomalie non è bloccante: il metodo ritornerà non appena l'autenticazione è avvenuta. Lancia un'eccezione nel caso in cui la chiave API non sia corretta, se la macchina o caratteristica non esistano, siano archiviate o siano disattivate.

4.3.1.3 IncomingDetection

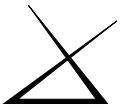
Questo record rappresenta una rilevazione in arrivo da processare.

Campi privati

- `String apiKey`: la chiave API della macchina che sta inviando la rilevazione;
- `String characteristic`: il nome della caratteristica rilevata;
- `double value`: il valore della rilevazione.

4.3.1.4 BusinessException

Vedi §4.3.2.2.



4.3.2 Gestione delle eccezioni

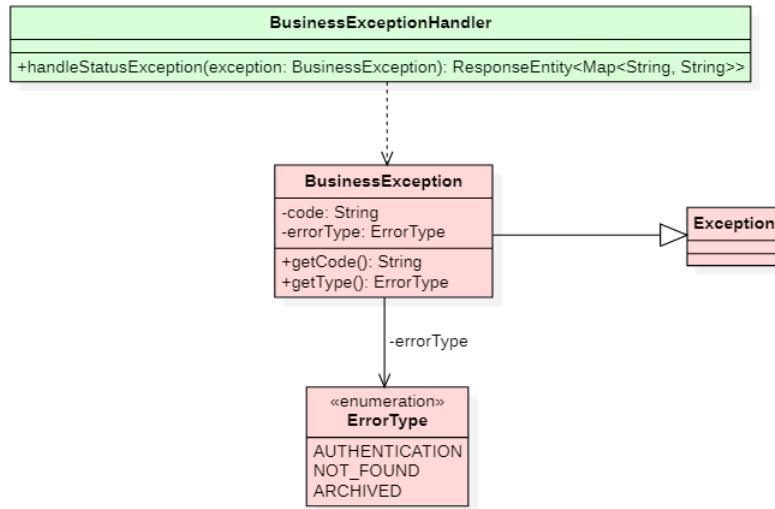


Figura 46: Diagramma delle classi relative a **BusinessException**.

Questo modulo si occupa di gestire gli errori dell'utente sotto forma di eccezioni, quindi di raccogliere la motivazione dell'errore e riportarla all'utente. Il ruolo principale è svolto da un **ExceptionHandler** di Spring Boot che cattura le eccezioni lanciate dal **RestController** e risponde di conseguenza.

4.3.2.1 BusinessExceptionHandler

Questa classe si occupa di catturare e gestire le eccezioni lanciate dal controller e produrre un adeguato messaggio di errore per il chiamante.

Annotazioni

- `@ControllerAdvice`.

Metodi

```
• @ExceptionHandler(BusinessException.class)
  ResponseEntity<Map<String, String>> handleStatusException(
```

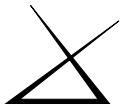
`BusinessException businessException:` l'eccezione lanciata dal controller.

)

Questo metodo viene chiamato quando il controller lancia un'eccezione. Una volta ricevuta, si occupa di convertirla in un messaggio di errore per il client, con annesso codice di stato HTTP.

4.3.2.2 BusinessException

Questa classe rappresenta un'eccezione derivante dalle componenti di business. Rappresenta un errore causato dall'esterno, e quindi gestibile.



Classe estesa

Questa classe estende `java.lang.Exception`.

Campi privati

- `String code`: il codice identificativo dell'errore;
- `ErrorType errorType`: il tipo o categoria di errore.

Metodi

- `String getCode()`
Getter che ritorna il codice identificativo dell'errore;
- `ErrorType getType()`
Getter che ritorna il tipo di errore.

4.3.2.3 ErrorType

Questa enumerazione rappresenta le possibili tipologie di errore.

Costanti

- `AUTHENTICATION`: caratterizza gli errori di autenticazione;
- `NOT_FOUND`: caratterizza gli errori dovuti a macchine o caratteristiche non esistenti;
- `ARCHIVED`: caratterizza gli errori dovuti a macchine o caratteristiche archiviate o disattivate.

4.3.3 Adapter

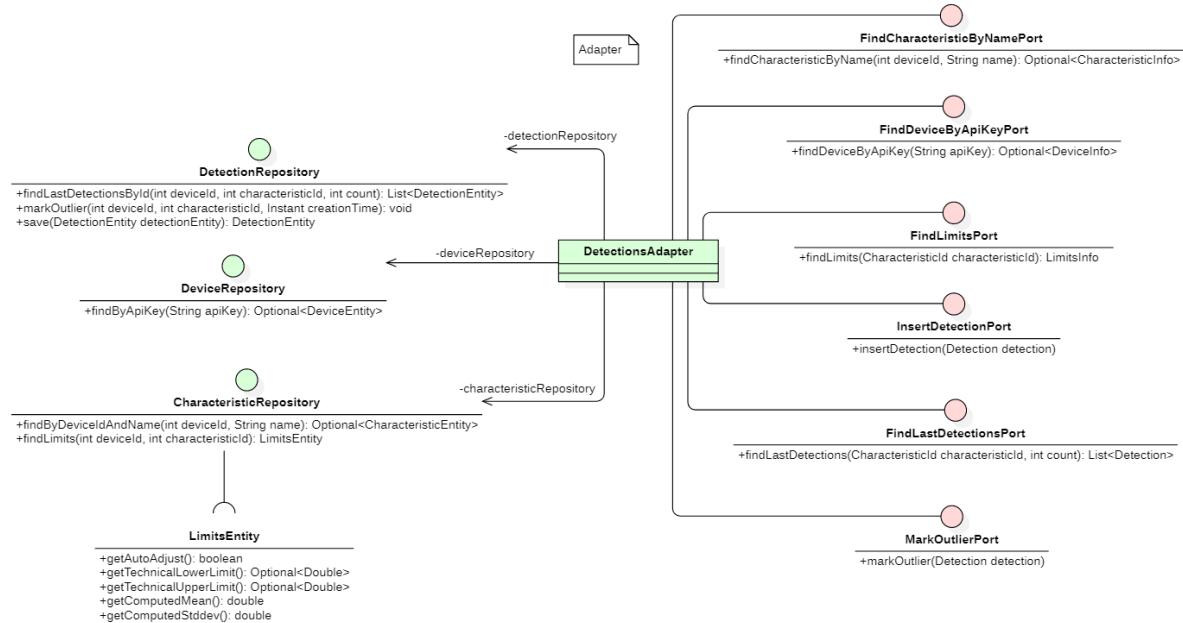
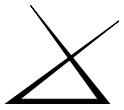


Figura 47: Diagramma delle classi relative a `DetectionsAdapter`.



Questo modulo si occupa di adattare le interfacce dei repository a quelle delle porte richieste dalla logica di business. Il ruolo principale è svolto dalla classe `DetectionsAdapter`, la quale adatta tutte i repository alle varie porte.

4.3.3.1 DetectionsAdapter

Questa classe si occupa di adattare i repository di Spring alle porte descritte dalla parte di business.

Annotazioni

- `@Component`.

Interfacce implementate

- `FindDeviceByApiKeyPort`;
- `FindCharacteristicByNamePort`;
- `FindLastDetectionsPort`;
- `InsertDetectionPort`;
- `FindLimitsPort`;
- `MarkOutlierPort`.

Campi privati

- `CharacteristicRepository characteristicRepository`: un'istanza del repository delle caratteristiche;
- `DeviceRepository deviceRepository`: un'istanza del repository delle macchine;
- `DetectionRepository detectionRepository`: un'istanza del repository delle rilevazioni.

Metodi

- `Optional<DeviceInfo> findDeviceByApiKey(`
 `String apiKey`: la chiave API della macchina da cercare, la quale potrebbe non esistere.
 `)`
 Implementazione dell'omonimo metodo definito in `FindDeviceByApiKeyPort`;
- `Optional<CharacteristicInfo> findCharacteristicByName(`
 `int deviceId`: l'identificativo della macchina, che deve esistere, a cui appartiene la caratteristica da cercare;
 `String name`: il nome della caratteristica da cercare.
 `)`
 Implementazione dell'omonimo metodo definito in `FindCharacteristicByNamePort`;
- `List<Detection> findLastDetections(`
 `CharacteristicId characteristicId`: l'identificativo globale della caratteristica a cui appartengono le rilevazioni da ottenere. Si può assumere che esista una caratteristica con tale identificativo;

```
    int count: il numero di rilevazioni da ottenere.  
)  
Implementazione dell'omonimo metodo definito in FindLastDetectionsPort;  
  
• void insertDetection(  
  
    Detection detection: la rilevazione da memorizzare.  
)  
Implementazione dell'omonimo metodo definito in InsertDetectionPort;  
  
• LimitsInfo findLimits(  
  
    CharacteristicId characteristicId: l'identificativo globale della caratteristica a cui  
    appartengono i limiti da ottenere. Si può assumere che esista una caratteristica con tale  
    identificativo.  
)  
Implementazione dell'omonimo metodo definito in FindLimitsPort;  
  
• void markOutlier(  
  
    Detection detection: la rilevazione da marcare come anomala.  
)  
Implementazione dell'omonimo metodo definito in MarkOutlierPort.
```

4.3.3.2 DetectionRepository

Questa interfaccia espone le query che è possibile effettuare sul database, in particolare sulla tabella detection, lasciando a Spring il compito d'implementarla.

Annotazioni

- @Repository.

Interfacce estese

- CrudRepository<DetectionEntity, DetectionEntityId>.

Metodi

- DetectionEntity save(

 DetectionEntity entity: l'entità della rilevazione da memorizzare.
)
Questo metodo è ereditato da CrudRepository e si occupa di persistere una rilevazione;
- @Query(value = "...", nativeQuery = true)
List<DetectionEntity> findLastDetectionsById(

 @Param("deviceId") int deviceId: l'identificativo della macchina;
 @Param("characteristicId") int characteristicId: l'identificativo della caratteristica all'interno della macchina;
 @Param("count") int count: il numero di rilevazioni da ottenere.

)

Questo metodo si occupa di ottenere le ultime `count` rilevazioni della caratteristica specificata dai primi due parametri. L'implementazione del parametro `value` dell'annotazione `@Query` è descritta nel paragrafo §5.3.5.1;

- `@Modifying
@Transactional
@Query(value = "...")
void markOutlier(

 @Param("deviceId") int deviceId: l'identificativo della macchina a cui appartiene la rilevazione;

 @Param("characteristicId") int characteristicId: l'identificativo della caratteristica all'interno della macchina a cui appartiene la rilevazione;

 @Param("creationTime") Instant creationTime: l'istante in cui è stata creata la rilevazione.

)`

Questo metodo si occupa di marcare come anomala la rilevazione identificata dai tre parametri. L'implementazione del parametro `value` dell'annotazione `@Query` è descritta nel paragrafo §5.3.5.2.

4.3.3.3 DeviceRepository

Questa interfaccia espone le query che è possibile effettuare sul database, in particolare sulla tabella `device`, lasciando a Spring il compito d'implementarla.

Annotazioni

- `@Repository`.

Interfacce estese

- `CrudRepository<DeviceEntity, Integer>`.

Metodi

- `Optional<DeviceEntity> findByApiKey(

 String apiKey: la chiave API della macchina da trovare.

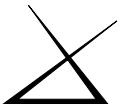
)`
- Questo metodo si occupa di trovare una macchina data la sua chiave API. Se esiste ritorna l'entità corrispondente, altrimenti ritorna `Optional.empty()`.

4.3.3.4 CharacteristicRepository

Questa interfaccia espone le query che è possibile effettuare sul database, in particolare sulla tabella `characteristic`, lasciando a Spring il compito d'implementarla.

Annotazioni

- `@Repository`.



Interfacce estese

- CrudRepository<CharacteristicEntity, CharacteristicEntityId>.

Metodi

- Optional<CharacteristicEntity> findByDeviceIdAndName(
 int deviceId: l'identificativo della macchina, che deve esistere, a cui appartiene la caratteristica da cercare;
 String name: il nome della caratteristica da cercare.
)
Questo metodo si occupa di cercare una caratteristica dato l'identificativo della sua macchina e il suo nome. Se esiste ritorna l'entità corrispondente, altrimenti ritorna Optional.empty();
- @Query(value = "...", nativeQuery = true)
LimitsEntity findLimits(
 @Param("deviceId") int deviceId: l'identificativo della macchina a cui appartiene la caratteristica;
 @Param("characteristicId") int characteristicId: l'identificativo della caratteristica all'interno della macchina.
)
Questo metodo si occupa di ottenere i limiti tecnici e di processo della caratteristica specificata dai suoi argomenti. L'implementazione del parametro value dell'annotazione @Query è descritta nel paragrafo §5.3.6.1.

4.3.3.5 LimitsEntity

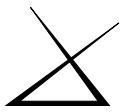
Questa interfaccia descrive il risultato della query definita nel metodo `findLimits` di `CharacteristicRepository`, lasciando a Spring il compito d'implementarla.

Metodi

- boolean getAutoAdjust()
Ritorna se l'auto-adjust è abilitato o meno;
- Optional<Double> getTechnicalLowerLimit()
Ritorna il valore del limite tecnico inferiore, se presente;
- Optional<Double> getTechnicalUpperLimit()
Ritorna il valore del limite tecnico superiore, se presente;
- double getComputedMean()
Ritorna il valore della media calcolata con l'auto-adjust;
- double getComputedStddev()
Ritorna il valore della deviazione standard calcolata con l'auto-adjust.

4.3.3.6 FindCharacteristicByNamePort

Questa interfaccia descrive l'abilità di poter cercare una caratteristica tra quelle memorizzate tramite il suo identificativo e permette di separare la sua implementazione dalla logica di business.



Metodi

- `Optional<CharacteristicInfo> findCharacteristicByName(`

`int deviceId:` l'identificativo della macchina, che deve esistere, a cui appartiene la caratteristica da cercare;

`String name:` il nome della caratteristica da cercare.

)

Questo metodo cerca i dettagli di una caratteristica dato l'identificativo della macchina a cui appartiene e il suo nome e se esiste li ritorna, altrimenti ritorna `Optional.empty()`.

4.3.3.7 FindDeviceByApiKeyPort

Questa interfaccia descrive l'abilità di poter cercare una macchina tra quelle memorizzate tramite la sua chiave API e permette di separare la sua implementazione dalla logica di business.

Metodi

- `Optional<DeviceInfo> findDeviceByApiKey(`

`String apiKey:` la chiave API della macchina da cercare. Potrebbe non esistere.

)

Questo metodo cerca i dettagli di una macchina e se esiste li ritorna, altrimenti ritorna `Optional.empty()`.

4.3.3.8 FindLimitsPort

Questa interfaccia descrive l'abilità di poter cercare i limiti tecnici e di processo di una caratteristica tramite il suo identificativo e permette di separare la sua implementazione dalla logica di business.

Metodi

- `LimitsInfo findLimits(`

`CharacteristicId characteristicId:` l'identificativo globale della caratteristica di cui ottenere i limiti, la quale deve esistere.

)

Questo metodo ritorna i limiti tecnici e di processo di una caratteristica.

4.3.3.9 InsertDetectionPort

Questa interfaccia descrive l'abilità di poter persistere una nuova rilevazione e permette di separare la sua implementazione dalla logica di business.

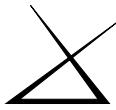
Metodi

- `void insertDetection(`

`Detection detection:` la rilevazione da persistere.

)

Questo metodo si occupa di persistere una rilevazione per poterla riottenere successivamente.



4.3.3.10 FindLastDetectionsPort

Questa interfaccia descrive l'abilità di poter ottenere le ultime rilevazioni di una caratteristica e permette di separare la sua implementazione dalla logica di business.

Metodi

- `List<Detection> findLastDetections()`

`CharacteristicId characteristicId`: l'identificativo globale della caratteristica di cui cercare le rilevazioni, la quale deve esistere;

`int count`: il numero massimo di rilevazioni da ottenere.

)

Questo metodo ritorna le ultime `count` rilevazioni della caratteristica specificata, o meno se non ne esistono abbastanza.

4.3.3.11 MarkOutlierPort

Questa interfaccia descrive l'abilità di poter marcare una rilevazione come anomala e permette di separare la sua implementazione dalla logica di business.

Metodi

- `void markOutlier()`

`Detection detection`: la rilevazione da marcare come anomala.

)

Questo metodo si occupa di marcare una rilevazione come anomala.

4.3.4 Service

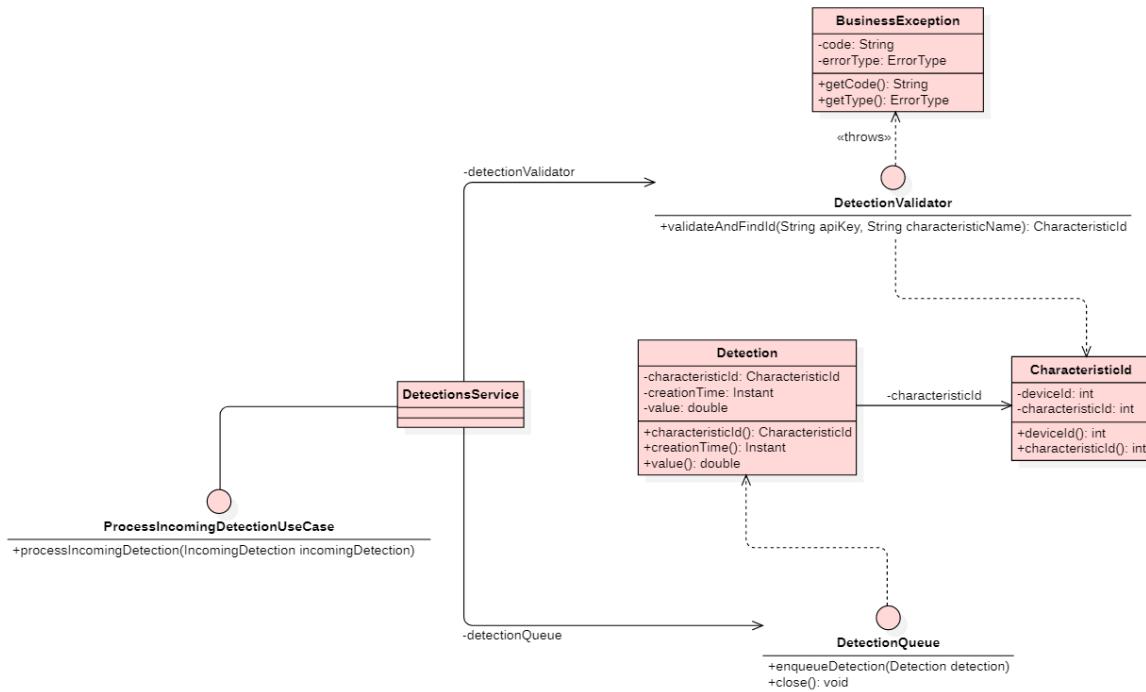


Figura 48: Diagramma delle classi relative a **DetectionsService**.

Questo modulo è l'entrata nella logica di business. Il ruolo principale è svolto dalla classe **DetectionsService**, la quale si occupa di dividere i compiti da svolgere tra le altre classi della logica di business.

4.3.4.1 DetectionsService

Questa classe si occupa di gestire le rilevazioni in arrivo da una macchina.

Interfacce implementate

- **ProcessIncomingDetectionUseCase**.

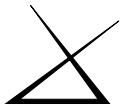
Campi privati

- **DetectionValidator detectionValidator**: un'istanza di **DetectionValidator** a cui delegare il compito di validare una rilevazione in arrivo;
- **DetectionQueue detectionQueue**: un'istanza di **DetectionQueue** a cui delegare il compito di processare le rilevazioni in background.

Metodi

- **void processIncomingDetection()**

IncomingDetection incomingDetection: la rilevazione in arrivo da processare.



```
) throws BusinessException
```

Questo metodo implementa l'omonimo metodo definito in `ProcessIncomingDetectionUseCase`.

4.3.4.2 DetectionValidator

Questa interfaccia descrive l'abilità di validare una rilevazione in arrivo da una macchina.

Metodi

- `CharacteristicId validateAndFindId(`

`String apiKey`: la chiave API fornita dalla macchina e da validare;

`String characteristicName`: il nome della caratteristica, all'interno della macchina, da validare.

```
) throws BusinessException
```

Questo metodo si occupa di validare una rilevazione in arrivo, in particolare valida la sua chiave API e il nome della caratteristica, controllando che esistano e che non siano archiviate/disattivate. In caso di successo ritorna l'identificativo globale della caratteristica, altrimenti lancia un'eccezione con la motivazione del fallimento.

4.3.4.3 DetectionQueue

Questa interfaccia descrive l'abilità di accodare una rilevazione per essere processata successivamente in background, senza quindi bloccare l'utilizzatore.

Metodi

- `void enqueueDetection(`

`Detection detection`: la rilevazione da accodare.

```
)
```

Questo metodo accoda una rilevazione per essere processata successivamente in background;

- `void close()`

Questo metodo permette di chiudere la coda, bloccando il chiamante finché tutte le rilevazioni non siano processate.

4.3.4.4 Detection

Questo record rappresenta una rilevazione.

Campi privati

- `CharacteristicId characteristicId`: l'identificativo globale della caratteristica a cui è associata la rilevazione;
- `Instant creationTime`: l'istante in cui è stata misurata la rilevazione;
- `double value`: il valore della rilevazione.

4.3.4.5 CharacteristicId

Questo record rappresenta l'identificativo globale di una caratteristica.

Campi privati

- `int deviceId`: l'identificativo della macchina a cui appartiene la caratteristica;
- `int characteristicId`: l'identificativo della caratteristica all'interno della macchina.

4.3.4.6 ProcessIncomingDetectionUseCase

Vedi §4.3.1.2.

4.3.4.7 BusinessException

Vedi §4.3.2.2.

4.3.5 Validator

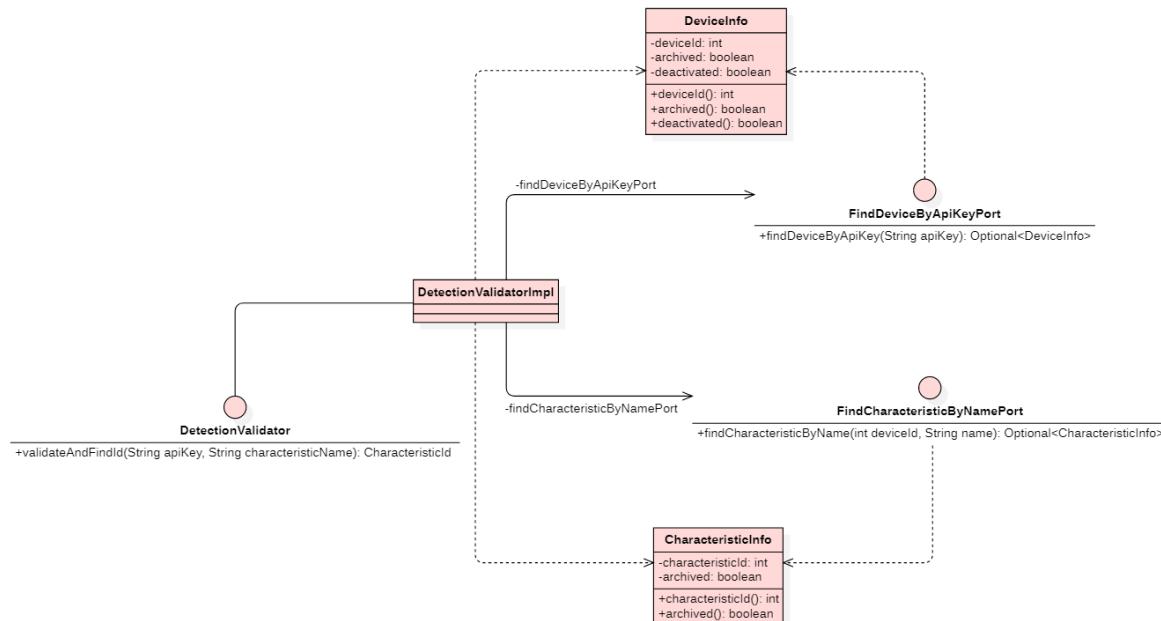


Figura 49: Diagramma delle classi relative a `DetectionValidator`.

Questo modulo si occupa di implementare la logica di business relativa alla validazione di una rilevazione in arrivo. Il ruolo principale è svolto dalla classe `DetectionValidatorImpl`, la quale implementa l'interfaccia `DetectionValidator` e utilizza le porte `FindDeviceByApiKeyPort` e `FindCharacteristicByNamePort` per ottenere le informazioni necessarie alla validazione.

4.3.5.1 DetectionValidatorImpl

Questa classe si occupa di validare le rilevazioni in arrivo da una macchina.

Interfacce implementate

- `DetectionValidator`.

Campi privati

- `FindDeviceByApiKeyPort findDeviceByApiKeyPort`: una porta necessaria per ottenere le informazioni di una macchina a partire dalla sua chiave API;
- `FindCharacteristicByNamePort findCharacteristicByNamePort`: una porta necessaria per ottenere le informazioni di una caratteristica a partire dall'identificativo della macchina a cui appartiene e il suo nome.

Metodi

- `CharacteristicId validateAndFindId(
 String apiKey: la chiave API fornita dalla macchina e da validare;
 String characteristicName: il nome della caratteristica, all'interno della macchina, da validare.
) throws BusinessException`
Questo metodo implementa l'omonimo metodo definito in `DetectionValidator`.

4.3.5.2 DeviceInfo

Questo record rappresenta le informazioni di una macchina utili alla validazione di una rilevazione.

Campi privati

- `int deviceId`: l'identificativo della macchina;
- `boolean archived`: `true` se la macchina è archiviata, `false` altrimenti;
- `boolean deactivated`: `true` se la macchina è disattivata, `false` altrimenti.

4.3.5.3 CharacteristicInfo

Questo record rappresenta le informazioni di una caratteristica utili alla validazione di una rilevazione.

Campi privati

- `int characteristicId`: l'identificativo della caratteristica all'interno della sua macchina;
- `boolean archived`: `true` se la caratteristica è archiviata, `false` altrimenti.

4.3.5.4 DetectionValidator

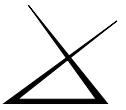
Vedi §4.3.4.2.

4.3.5.5 FindDeviceByApiKeyPort

Vedi §4.3.3.7.

4.3.5.6 FindCharacteristicByNamePort

Vedi §4.3.3.6.



4.3.6 Queue

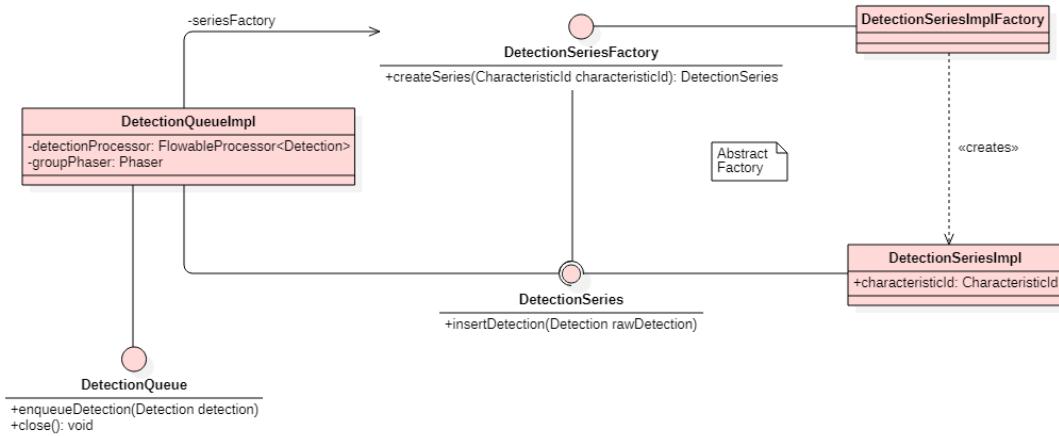


Figura 50: Diagramma delle classi relative a `DetectionQueue`.

Questo modulo si occupa di processare le rilevazioni in background. Il ruolo principale è svolto dalla classe `DetectionQueueImpl`, la quale raggruppa le rilevazioni in base alla loro caratteristica e processa ogni caratteristica in parallelo, ma sequenzialmente al suo interno, per mantenere l'ordine delle rilevazioni. La gestione delle singole caratteristiche è poi lasciato a delle `DetectionSeries`, le quali vengono create da delle `DetectionSeriesFactory`.

4.3.6.1 `DetectionQueueImpl`

Questa classe si occupa di accodare una rilevazione per essere processata successivamente in background, senza quindi bloccare l'utilizzatore.

Interfacce implementate

- `DetectionQueue`.

Campi privati

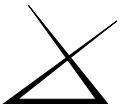
- `DetectionSeriesFactory seriesFactory`: la factory per creare una `DetectionSeries` per ogni caratteristica in coda;
- `FlowableProcessor<Detection> detectionProcessor`: un canale di RxJava 3 che si occupa di raggruppare le rilevazioni e processarle in differenti thread;
- `Phaser groupPhaser`: una primitiva di sincronizzazione che coordina la chiusura della coda.

Costruttori

- `DetectionQueueImpl(`

`DetectionSeriesFactory seriesFactory`: il valore per il campo `seriesFactory`.

`).`



Metodi

- `void enqueueDetection()`

`Detection detection:` la rilevazione da accodare.

)

Questo metodo implementa l'omonimo metodo definito in `DetectionQueue`;

- `void close()`

Questo metodo implementa l'omonimo metodo definito in `DetectionQueue`.

4.3.6.2 DetectionSeriesFactory

Questa interfaccia descrive l'abilità di creare una nuova `DetectionSeries` per una data caratteristica.

Metodi

- `DetectionSeries createSeries()`

`CharacteristicId characteristicId:` l'identificativo globale della caratteristica a cui sarà associata la `DetectionSeries`.

)

Questo metodo crea una nuova `DetectionSeries` per una data caratteristica.

4.3.6.3 DetectionSeries

Questa interfaccia descrive l'abilità di processare una serie di rilevazioni, tutte relative a una singola caratteristica.

Metodi

- `void insertDetection()`

`Detection rawDetection:` la rilevazione da processare. Deve appartenere alla caratteristica associata a questa serie.

)

Questo metodo inserisce una nuova rilevazione nella serie, processandola.

4.3.6.4 DetectionSeriesImplFactory

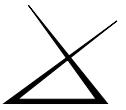
Questa classe si occupa di creare una `DetectionSeriesImpl`.

Interfacce implementate

- `DetectionSeriesFactory`.

Campi privati

- `SeriesPortFacade ports:` le porte utilizzate da `DetectionSeriesImpl`;
- `ControlLimitsCalculator controlLimitsCalculator:` un calcolatore di limiti di controllo utilizzato dalla `DetectionSeriesImpl`;
- `ControlChartsGroup controlChartsGroup:` le carte di controllo che la `DetectionSeriesImpl` applicherà.



Metodi

- `DetectionSeries createSeries(`

`CharacteristicId characteristicId:` l'identificativo globale della caratteristica a cui sarà associata la `DetectionSeries`.

)

Questo metodo implementa l'omonimo metodo definito in `DetectionSeriesFactory`, creando una `DetectionSeriesImpl`.

4.3.6.5 DetectionQueue

Vedi §4.3.4.3.

4.3.6.6 DetectionSeriesImpl

Vedi §4.3.7.1.

4.3.7 Series

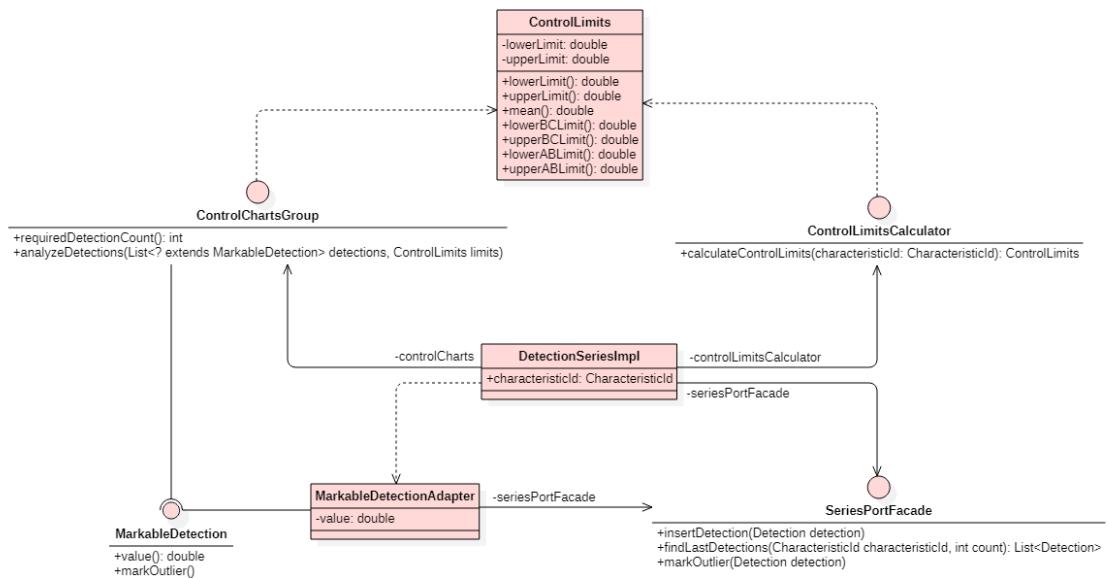
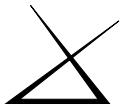


Figura 51: Diagramma delle classi relative a `DetectionSeries`.

Questo modulo si occupa di gestire una serie di rilevazioni appartenenti a una caratteristica. Il ruolo principale è svolto dalla classe `DetectionSeriesImpl`, la quale si occupa di organizzare:

- la persistenza delle rilevazioni;
- l'ottenimento delle informazioni necessarie alle carte di controllo, quali:
 - i limiti di controllo;
 - le ultime rilevazioni.



- l'applicazione delle carte di controllo;
 - l'aggiornamento delle rilevazioni come anomalie.

La loro implementazione viene però delegata ad altre classi.

4.3.7.1 DetectionSeriesImpl

Questa classe si occupa di processare una serie di rilevazioni relative a una stessa caratteristica.

Interfacce implementate

- DetectionSeries.

Campi privati

- **SeriesPortFacade ports**: le porte utilizzate da questa classe;
 - **ControlLimitsCalculator controlLimitsCalculator**: un calcolatore di limiti di controllo;
 - **ControlChartsGroup controlChartsGroup**: le carte di controllo da applicare;
 - **CharacteristicId characteristicId**: l'identificativo globale della caratteristica associata a questa serie.

Metodi

- void insertDetection()

Detection rawDetection: la rilevazione da processare. Deve appartenere alla caratteristica associata a questa serie.

)

Questo metodo implementa l'omonimo metodo definito in `DetectionSeries`.

4.3.7.2 ControlLimitsCalculator

Questa interfaccia si occupa di calcolare i limiti di controllo di una caratteristica.

Metodi

- ControlLimits calculateControlLimits(

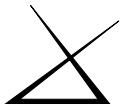
CharacteristicId characteristicId: l'identificativo globale della caratteristica di cui calcolare i limiti di controllo.

)

Questo metodo si occupa di calcolare i limiti di controllo di una caratteristica, ritornandoli.

4.3.7.3 SeriesPortFacade

Questa interfaccia raccoglie insieme l'interfaccia di alcune porte che sono usate sempre insieme.



Metodi

- `void insertDetection()`

`Detection detection`: la rilevazione da persistere.

)

Questo metodo è equivalente all'omonimo definito in `InsertDetectionPort`;

- `List<Detection> findLastDetections()`

`CharacteristicId characteristicId`: l'identificativo globale della caratteristica di cui cercare le rilevazioni, la quale deve esistere;

`int count`: il numero massimo di rilevazioni da ottenere.

)

Questo metodo è equivalente all'omonimo definito in `FindLastDetectionsPort`;

- `void markOutlier()`

`Detection detection`: la rilevazione da marcare come anomala.

)

Questo metodo è equivalente all'omonimo definito in `MarkOutlierPort`.

4.3.7.4 ControlChartsGroup

Questa interfaccia rappresenta un insieme di carte di controllo, il quale è equivalente a una singola carta di controllo che applica tutte le regole di quelle presenti nell'insieme.

Interfacce estese

- `ControlChart`.

4.3.7.5 MarkableDetection

Questa interfaccia rappresenta una rilevazione che può essere marcata come anomala.

Metodi

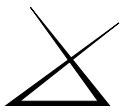
- `double value()`
ritorna il valore della rilevazione;
- `void markOutlier()`
marca la rilevazione come anomala.

4.3.7.6 MarkableDetectionAdapter

Questa classe implementa una rilevazione che può essere marcata come anomala.

Interfacce implementate

- `MarkableDetection`.



Campi privati

- `SeriesPortFacade seriesPortFacade`: la porta utilizzata per marcare le rilevazioni come anomale;
- `Detection detection`: la rilevazione.

Metodi

- `double value()`
Questo metodo implementa l'omonimo definito in `MarkableDetection`;
- `void markOutlier()`
Questo metodo implementa l'omonimo definito in `MarkableDetection`.

4.3.7.7 ControlLimits

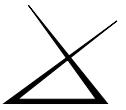
Questa record rappresenta dei limiti di controllo.

Campi privati

- `double lowerLimit`: il limite di controllo inferiore;
- `double upperLimit`: il limite di controllo superiore.

Metodi

- `double mean()`
Ritorna la media dei limiti;
- `double lowerBCLimit()`
Ritorna il valore limite tra la zona B e la zona C inferiori;
- `double upperBCLimit()`
Ritorna il valore limite tra la zona B e la zona C superiori;
- `double lowerABLimit()`
Ritorna il valore limite tra la zona A e la zona B inferiori;
- `double upperABLimit()`
Ritorna il valore limite tra la zona A e la zona B superiori.



4.3.8 SeriesPortFacade

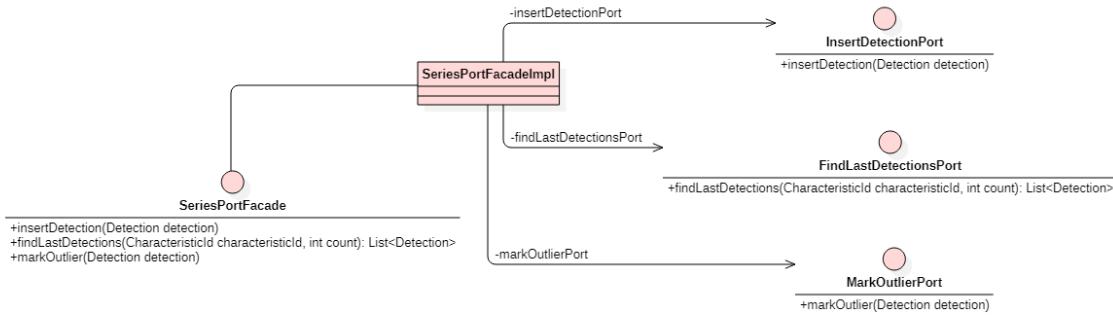


Figura 52: Diagramma delle classi relative a `SeriesPortFacade`.

Questo modulo si occupa di semplificare l'interfaccia di `DetectionSeriesImpl` raggruppando delle porte usate sempre insieme. Il ruolo principale è svolto dalla classe `SeriesPortFacadeImpl` che implementa tale raggruppamento delegando alle vere implementazioni di tali porte.

4.3.8.1 SeriesPortFacadeImpl

Questa classe implementa l'interfaccia di alcune porte che sono usate sempre insieme.

Interfacce implementate

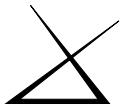
- `SeriesPortFacade`.

Campi privati

- `InsertDetectionPort insertDetectionPort`: la porta a cui viene inoltrato `insertDetection`;
- `FindLastDetectionsPort findLastDetectionsPort`: la porta a cui viene inoltrato `findLastDetections`;
- `MarkOutlierPort markOutlierPort`: la porta a cui viene inoltrato `markOutlier`.

Metodi

- `void insertDetection()`
 `Detection detection`: la rilevazione da persistere.
)
 Implementazione dell'omonimo metodo definito in `SeriesPortFacade`;
- `List<Detection> findLastDetections()`
 `CharacteristicId characteristicId`: l'identificativo globale della caratteristica di cui cercare le rilevazioni, la quale deve esistere;
 `int count`: il numero massimo di rilevazioni da ottenere.



)
Implementazione dell'omonimo metodo definito in **SeriesPortFacade**;

- **void markOutlier()**

Detection detection: la rilevazione da marcare come anomala.

)
Implementazione dell'omonimo metodo definito in **SeriesPortFacade**.

4.3.8.2 SeriesPortFacade

Vedi §4.3.7.3.

4.3.8.3 InsertDetectionPort

Vedi §4.3.3.9.

4.3.8.4 FindLastDetectionsPort

Vedi §4.3.3.10.

4.3.8.5 MarkOutlierPort

Vedi §4.3.3.11.

4.3.9 ControlLimitsCalculator

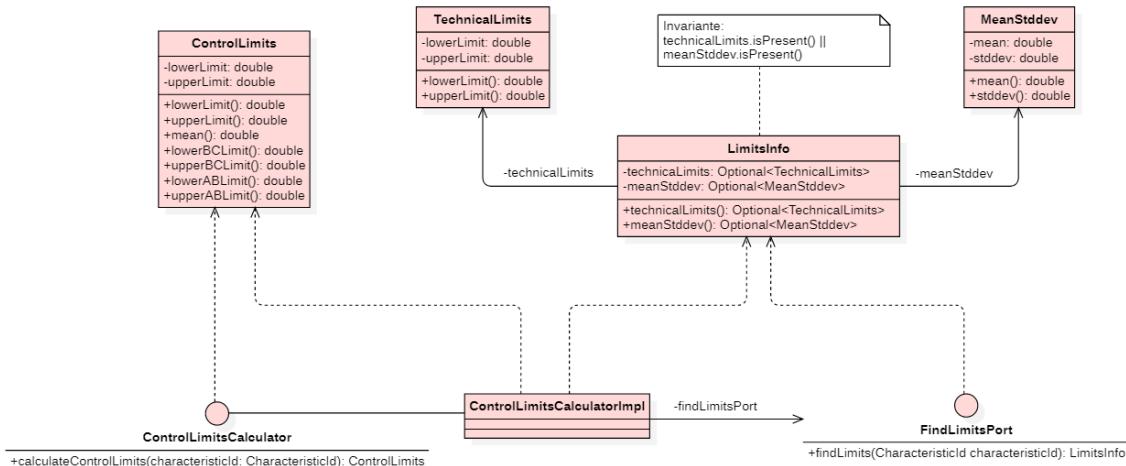
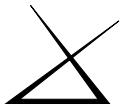


Figura 53: Diagramma delle classi relative a **ControlLimitsCalculator**.

Questo modulo si occupa del calcolo dei limiti di controllo. Il ruolo principale è svolto dalla classe **ControlLimitsCalculatorImpl**, che implementa tale calcolo a partire dai limiti tecnici e di processo, se presenti, ottenuti tramite la porta **FindLimitsPort**.



4.3.9.1 ControlLimitsCalculatorImpl

Questa classe si occupa di calcolare i limiti di controllo di una caratteristica.

Interfacce implementate

- `ControlLimitsCalculator`.

Campi privati

- `FindLimitsPort findLimitsPort`: la porta per ottenere i limiti tecnici e di processo di una caratteristica.

Metodi

- `ControlLimits calculateControlLimits(`

`CharacteristicId characteristicId`: l'identificativo globale della caratteristica di cui calcolare i limiti di controllo.

)

Implementazione dell'omonimo metodo definito in `ControlLimitsCalculator`.

4.3.9.2 LimitsInfo

Questo record rappresenta i limiti tecnici e di processo di una caratteristica, se presenti. Almeno uno dei due tipi di limiti deve essere presente.

Campi privati

- `Optional<TechnicalLimits> technicalLimits`: i limiti tecnici, se presenti;
- `Optional<MeanStddev> meanStddev`: i limiti di processo, se presenti.

4.3.9.3 TechnicalLimits

Questo record rappresenta i limiti tecnici di una caratteristica.

Campi privati

- `double lowerLimit`: il limite tecnico inferiore;
- `double upperLimit`: il limite tecnico superiore.

4.3.9.4 MeanStddev

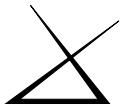
Questo record rappresenta i limiti di processo di una caratteristica.

Campi privati

- `double mean`: la media calcolata dall'auto-adjust;
- `double stddev`: la deviazione standard calcolata dall'auto-adjust.

4.3.9.5 ControlLimitsCalculator

Vedi §4.3.7.2.



4.3.9.6 ControlLimits

Vedi §4.3.7.7.

4.3.9.7 FindLimitsPort

Vedi §4.3.3.8.

4.3.10 ControlChartsGroup

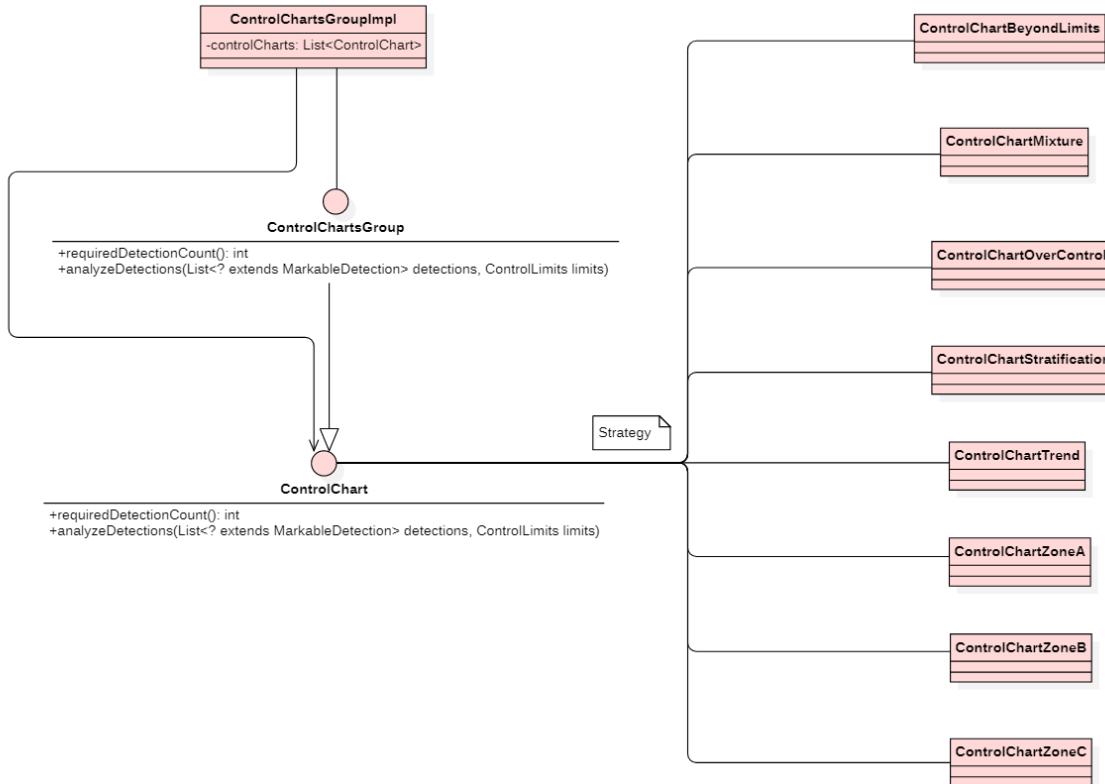


Figura 54: Diagramma delle classi relative a **ControlChart**.

Questo modulo si occupa di implementare e organizzare le carte di controllo. Il ruolo principale è svolto dalle varie implementazioni delle carte di controllo, che vengono poi raggruppate insieme dalla classe **ControlChartsGroupImpl**.

4.3.10.1 ControlChartsGroupImpl

Questa classe implementa un insieme di carte di controllo.

Interfacce implementate

- **ControlChartsGroup**.

Campi privati

- `List<? extends ControlChart> controlCharts`: una lista di carte di controllo.

Metodi

- `int requiredDetectionCount()`
Implementazione dell'omonimo metodo definito in `ControlChartsGroup`;
- `void analyzeDetections()`

`List<? extends MarkableDetection> lastDetections`: le rilevazioni da analizzare;
`ControllLimits limits`: i limiti di controllo della caratteristica a cui appartengono le rilevazioni.

)
Implementazione dell'omonimo metodo definito in `ControlChartsGroup`.

4.3.10.2 ControlChart

Questa interfaccia descrive l'abilità di analizzare una serie di rilevazioni per trovarne di anomale.

Metodi

- `int requiredDetectionCount()`
Ritorna il numero di rilevazioni richieste per l'analisi. Questo metodo deve ritornare sempre lo stesso valore per una data istanza;
- `void analyzeDetections()`

`List<? extends MarkableDetection> lastDetections`: le rilevazioni da analizzare. Deve avere la lunghezza specificata da `requiredDetectionCount`;
`ControllLimits limits`: i limiti di controllo della caratteristica a cui appartengono le rilevazioni.

)
Analizza le rilevazioni alla luce dei limiti di controllo e marca quelle anomale.

4.3.10.3 ControlChartBeyondLimits

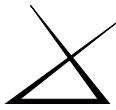
Questa classe implementa la carta di controllo che identifica i punti oltre i limiti di controllo.

Metodi

- `int requiredDetectionCount()`
Implementazione dell'omonimo metodo definito in `ControlChart`;
- `void analyzeDetections()`

`List<? extends MarkableDetection> lastDetections`: le rilevazioni da analizzare;
`ControllLimits limits`: i limiti di controllo della caratteristica a cui appartengono le rilevazioni.

)
Implementazione dell'omonimo metodo definito in `ControlChart`.



4.3.10.4 ControlChartMixture

Questa classe implementa la carta di controllo che identifica se 8 punti consecutivi non appartengono alle zone C.

Metodi

- `int requiredDetectionCount()`
Implementazione dell'omonimo metodo definito in `ControlChart`;
- `void analyzeDetections(`
`List<? extends MarkableDetection> lastDetections:` le rilevazioni da analizzare;
`ControllLimits limits:` i limiti di controllo della caratteristica a cui appartengono le rilevazioni.
)
Implementazione dell'omonimo metodo definito in `ControlChart`.

4.3.10.5 ControlChartOverControl

Questa classe implementa la carta di controllo che identifica se 14 punti consecutivi sono a zig-zag.

Metodi

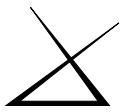
- `int requiredDetectionCount()`
Implementazione dell'omonimo metodo definito in `ControlChart`;
- `void analyzeDetections(`
`List<? extends MarkableDetection> lastDetections:` le rilevazioni da analizzare;
`ControllLimits limits:` i limiti di controllo della caratteristica a cui appartengono le rilevazioni.
)
Implementazione dell'omonimo metodo definito in `ControlChart`.

4.3.10.6 ControlChartStratification

Questa classe implementa la carta di controllo che identifica se 15 punti consecutivi appartengono alle zone C.

Metodi

- `int requiredDetectionCount()`
Implementazione dell'omonimo metodo definito in `ControlChart`;
- `void analyzeDetections(`
`List<? extends MarkableDetection> lastDetections:` le rilevazioni da analizzare;
`ControllLimits limits:` i limiti di controllo della caratteristica a cui appartengono le rilevazioni.
)
Implementazione dell'omonimo metodo definito in `ControlChart`.



4.3.10.7 ControlChartTrend

Questa classe implementa la carta di controllo che identifica se 7 punti consecutivi seguono lo stesso ordine.

Metodi

- `int requiredDetectionCount()`
Implementazione dell'omonimo metodo definito in `ControlChart`;
- `void analyzeDetections()`

`List<? extends MarkableDetection> lastDetections:` le rilevazioni da analizzare;
`Controllimits limits:` i limiti di controllo della caratteristica a cui appartengono le rilevazioni.

)
Implementazione dell'omonimo metodo definito in `ControlChart`.

4.3.10.8 ControlChartZoneA

Questa classe implementa la carta di controllo che identifica se 2 punti su 3 consecutivi sono all'interno di una delle due zone A o oltre.

Metodi

- `int requiredDetectionCount()`
Implementazione dell'omonimo metodo definito in `ControlChart`;
- `void analyzeDetections()`

`List<? extends MarkableDetection> lastDetections:` le rilevazioni da analizzare;
`Controllimits limits:` i limiti di controllo della caratteristica a cui appartengono le rilevazioni.

)
Implementazione dell'omonimo metodo definito in `ControlChart`.

4.3.10.9 ControlChartZoneB

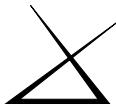
Questa classe implementa la carta di controllo che identifica se 4 punti su 5 consecutivi sono all'interno di una delle due zone B o oltre.

Metodi

- `int requiredDetectionCount()`
Implementazione dell'omonimo metodo definito in `ControlChart`;
- `void analyzeDetections()`

`List<? extends MarkableDetection> lastDetections:` le rilevazioni da analizzare;
`Controllimits limits:` i limiti di controllo della caratteristica a cui appartengono le rilevazioni.

)
Implementazione dell'omonimo metodo definito in `ControlChart`.



4.3.10.10 ControlChartZoneC

Questa classe implementa la carta di controllo che identifica se 7 punti consecutivi sono dallo stesso lato rispetto alla media.

Metodi

- `int requiredDetectionCount()`
Implementazione dell'omonimo metodo definito in `ControlChart`;
- `void analyzeDetections(`
`List<? extends MarkableDetection> lastDetections:` le rilevazioni da analizzare;
`ControllLimits limits:` i limiti di controllo della caratteristica a cui appartengono le rilevazioni.
`)`
Implementazione dell'omonimo metodo definito in `ControlChart`.

4.3.10.11 ControlChartUtils

Questa classe implementa dei metodi di supporto per le carte di controllo.

Metodi

- `<T> Stream<List<T>> windows(`
`List<T> list:` la lista sorgente di valori;
`int size:` la dimensione delle liste nello `Stream` ritornato.
`)`
Questo metodo ritorna uno `Stream` di “finestre” della lista sorgente, cioè uno `Stream` di tutte le liste contenenti valori adiacenti nella lista sorgente;
- `void markAll(`
`List<? extends MarkableDetection> detections:` una lista di rilevazioni marcabili.
`)`
Questo metodo marca tutte le rilevazioni della lista.

4.3.10.12 ControlChartsGroup

Vedi §4.3.7.4.

4.3.11 Configuration

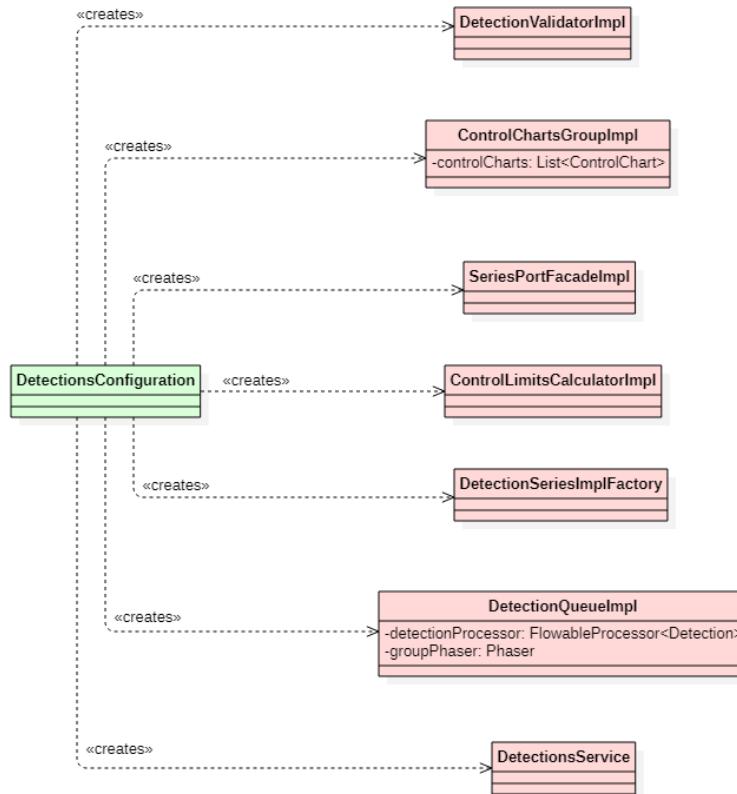


Figura 55: Diagramma delle classi relative a `DetectionsConfiguration`.

Questo modulo si occupa di gestire la dependency injection delle classi di business, le quali non potendo dipendere dal framework Spring Boot non possono godere della dependency injection automatica. Il ruolo principale è svolto dalla classe `DetectionsConfiguration`, la quale descrive a Spring Boot come creare istanze delle interfacce utilizzate all'interno della logica di business.

4.3.11.1 DetectionsConfiguration

Questa classe implementa una configurazione di Spring che descrive come creare le varie classi di business.

Annotazioni

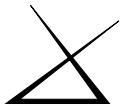
- `@Component`.

Metodi

- `@Bean`

```
DetectionValidator createDetectionValidator()
```

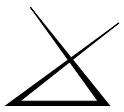
`FindDeviceByApiKeyPort findDeviceByApiKeyPort:` la `FindDeviceByApiKeyPort` da usare nel costruttore di `DetectionValidatorImpl`;



```
FindCharacteristicPort findCharacteristicPort: la FindCharacteristicPort da usare nel costruttore di DetectionValidatorImpl.  
)  
Questo metodo crea un'istanza di DetectionValidator;  


- @Bean  
ControlChartsGroup createControlCharts()  
Questo metodo crea un'istanza di ControlChartsGroup;
- @Bean  
SeriesPortFacade createSeriesFacadePort(  
  
InsertDetectionPort insertDetectionPort: la InsertDetectionPort da usare nel costruttore di SeriesPortFacadeImpl;  
FindLastDetectionsPort findLastDetectionsPort: la FindLastDetectionsPort da usare nel costruttore di SeriesPortFacadeImpl;  
MarkOutlierPort markOutlierPort: la MarkOutlierPort da usare nel costruttore di SeriesPortFacadeImpl.  
)  
Questo metodo crea un'istanza di SeriesPortFacade;
- @Bean  
ControlLimitsCalculator controlLimitsCalculator(  
  
FindLimitsPort findLimitsPort: la FindLimitsPort da usare nel costruttore di ControlLimitsCalculatorImpl.  
)  
Questo metodo crea un'istanza di ControlLimitsCalculator;
- @Bean  
DetectionSeriesFactory createDetectionSeriesFactory(  
  
SeriesPortFacade seriesPortFacade: la SeriesPortFacade da usare nel costruttore di DetectionSeriesImplFactory;  
ControlLimitsCalculator controlLimitsCalculator: il ControlLimitsCalculator da usare nel costruttore di DetectionSeriesImplFactory;  
ControlChartsGroup controlChartsGroup: il ControlChartsGroup da usare nel costruttore di DetectionSeriesImplFactory.  
)  
Questo metodo crea un'istanza di DetectionSeriesFactory;
- @Bean  
@Scope("singleton")  
DetectionQueue createDetectionQueue(  
  
DetectionSeriesFactory detectionSeriesFactory: la DetectionSeriesFactory da usare nel costruttore di DetectionQueueImpl.  
)  
Questo metodo crea un'istanza di DetectionQueue;

```



- @Bean
ProcessIncomingDetectionUseCase createProcessDetectionUseCase(
 DetectionValidator detectionValidation: il DetectionValidator da usare nel costruttore di DetectionsService;
 DetectionQueue detectionQueue: la DetectionQueue da usare nel costruttore di DetectionsService.
)
Questo metodo crea un'istanza di ProcessIncomingDetectionUseCase.

4.3.11.2 DetectionValidatorImpl

Vedi §4.3.5.1.

4.3.11.3 ControlChartsGroupImpl

Vedi §4.3.10.1.

4.3.11.4 SeriesPortFacadeImpl

Vedi §4.3.8.1.

4.3.11.5 ControlLimitsCalculatorImpl

Vedi §4.3.9.1.

4.3.11.6 DetectionSeriesImplFactory

Vedi §4.3.6.4.

4.3.11.7 DetectionQueueImpl

Vedi §4.3.6.1.

4.3.11.8 DetectionsService

Vedi §4.3.4.1.

5 Progettazione di dettaglio

5.1 Front-end UI

Le azioni hanno tutte origine da un'interazione di un utente con la pagina HTML.

5.1.1 LoginComponent

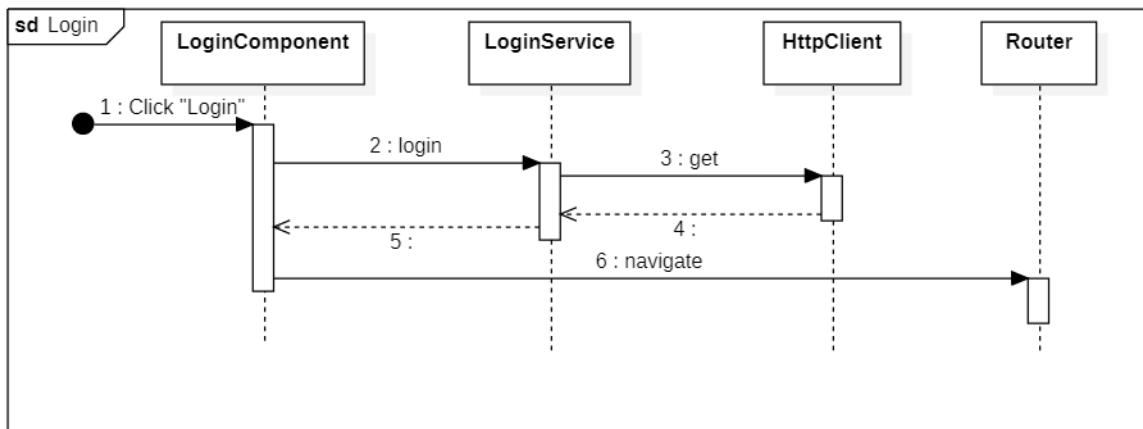


Figura 56: Diagramma di sequenza del processo di autenticazione.

1. l'utente clicca sul pulsante di login;
2. viene chiamato il servizio di login, passando nome utente e password;
3. il servizio effettua una chiamata HTTP GET attraverso il metodo apposito di HttpClient;
4. viene resituito un Observable al contesto LoginComponent;
5. in caso di successo, viene effettuata la navigazione alla pagina di monitoraggio rilevazioni, tramite la classe Router.

5.1.1.1 ngOnInit

Ereditato dall'interfaccia `OnInit`. Verifica la presenza del cookie PRODULYTICS_RM. In caso positivo, naviga alla pagina di monitoraggio rilevazioni tramite un oggetto della classe Router.

5.1.1.2 onSubmit

Nel caso in cui tutti gli input inseriti dall'utente rispettino le validazioni, esegue un tentativo di login. In caso di successo, naviga alla pagina di monitoraggio rilevazioni; altrimenti mostra un errore.

5.1.2 ToolbarComponent

5.1.2.1 openPwDialog

Apre la finestra di dialogo dedicata alla modifica della password (`ModifyPwComponent`).

5.1.2.2 isLoggedIn

Tramite un servizio che implementa `LoginAbstractService`, ritorna `true` se l'utente è autenticato; `false` altrimenti. Serve a determinare quando mostrare la barra di navigazione, in quanto non deve comparire a un utente non autenticato.

5.1.2.3 isAdmin

Tramite un servizio che implementa `LoginAbstractService`, ritorna `true` se l'utente ha permessi di amministratore; `false` altrimenti. Serve a determinare quando mostrare le opzioni per navigare alle pagine di gestione macchine e gestione utenti, disponibili solo agli amministratori.

5.1.2.4 getUsername

Tramite un servizio che implementa `LoginAbstractService`, ritorna una `string` con il nome dell'utente. Serve a mostrare il nome utente nella barra di navigazione.

5.1.2.5 logout

Tramite un servizio che implementa `LoginAbstractService`, effettua il logout, rimuovendo tutti i cookie di sessione e tutti gli elementi salvati in `localStorage`.

5.1.3 ModifyPwComponent

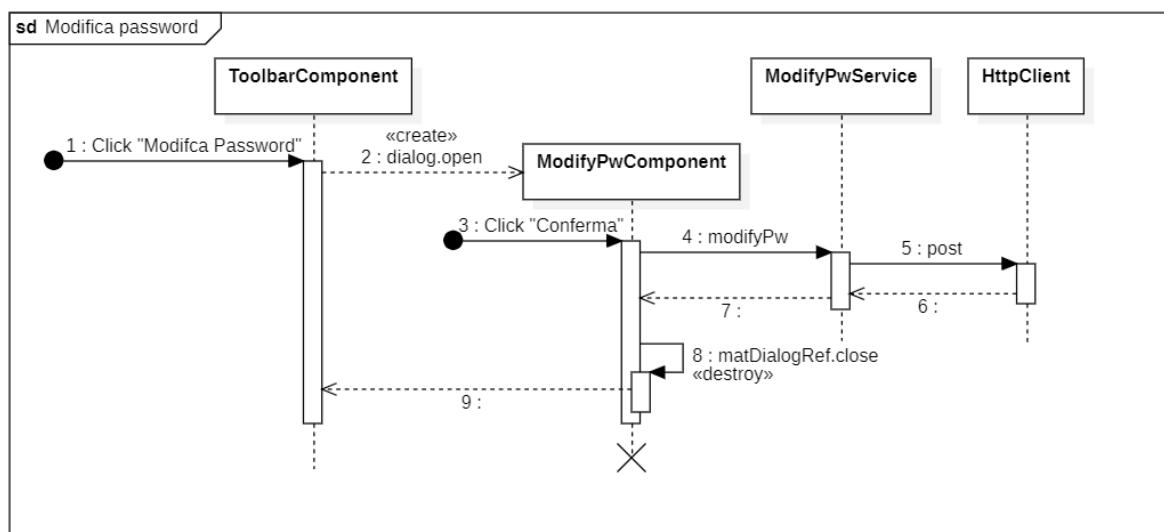
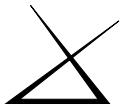


Figura 57: Diagramma di sequenza della modifica della password.

1. a partire dal menù utente della barra di navigazione, l'utente clicca sul pulsante di modifica password;
2. viene creata una finestra di dialogo tramite il metodo `open` dell'oggetto `dialog: MatDialog` nel quale l'utente può inserire i dati necessari al cambio password;
3. l'utente clicca sul tasto di conferma;



4. viene chiamato il servizio di modifica password, passando password corrente e nuova;
5. il servizio effettua una chiamata HTTP PUT attraverso il metodo apposito di `HttpClient`;
6. viene restituito un `Observable` al contesto `ModifyPwComponent`;
7. la finestra di dialogo viene chiusa tramite il metodo `close` dell'oggetto `matDialogRef: MatDialogRef`, passando i dati ottenuti al punto precedente al contesto `ToolbarComponent`;
8. l'utente viene notificato del successo o meno dell'operazione tramite l'oggetto `matSnackBar: MatSnackBar`.

5.1.3.1 `checkPasswords`

Parametri:

- `control: AbstractControl` - permette di ottenere i valori dei dati inseriti dall'utente.

Valida i dati inseriti dall'utente. In particolare, controlla che:

- la nuova password e la sua ripetizione siano uguali;
- la nuova password sia diversa da quella corrente.

Se almeno uno di questi controlli non è passato, vengono restituiti degli errori appropriati: `mismatch` nel primo caso, `mustBeDifferent` nel secondo.

5.1.3.2 `confirm`

Tramite un servizio che implementa `ModifyPwAbstractService`, effettua un tentativo di cambio password. In caso di successo, viene visualizzato un messaggio che conferma l'esito positivo all'utente, altrimenti viene visualizzato l'errore appropriato. Questi messaggi sono mostrati tramite `MatSnackBar`.

5.1.3.3 `cancel`

Annulla l'operazione, chiudendo la finestra di dialogo.

5.1.4 SelectionComponent

5.1.4.1 `hasChildren`

Parametri:

- `_index: number` - l'indice del nodo;
- `node: SelectionNnode` - il nodo.

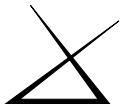
Ritorna `false` se il nodo passato come parametro è una foglia; `true` altrimenti.

5.1.4.2 `nodeIsChecked`

Parametri:

- `node: CharacteristicNode` - il nodo da controllare.

Ritorna `true` se il nodo passato come parametro è selezionato; `false` altrimenti.



5.1.4.3 toggleNodeCheck

Parametri:

- `checked: boolean` - rappresenta lo stato del nodo passato come parametro: `true` se è selezionato; `false` altrimenti;
- `node: CharacteristicNode` - il nodo.

Seleziona il nodo se `checked` è `false`; lo deselectiona altrimenti.

5.1.4.4 notifyChange

Metodo chiamato quando viene premuto il tasto di conferma dell'albero. Copia il valore di `checkedNodes` tramite `slice()` e lo assegna a `_checkedNodes`.

5.1.5 SelectionDataSource

5.1.5.1 connect

Ereditato da `DataSource`. Fornisce la sorgente dati che notifica i cambiamenti. Restituisce un `Observable<SelectionNode[]>`.

5.1.5.2 disconnect

Ereditato da `DataSource`. Termina il flusso di dati. Essendo una sorgente semplice, non è necessario fare nulla.

5.1.5.3 isNodeLoading

Parametri:

- `node: SelectionNode` - il nodo.

Restituisce `true` se il nodo passato come parametro è in fase di caricamento; `false` altrimenti.

5.1.5.4 handleChange

Parametri:

- `change: SelectionChange<SelectionNode>` - rappresenta l'aggiunta o la rimozione di uno o più nodi.

A seconda del valore di `change`, chiama `addCharacteristics` o `removeCharacteristics`.

5.1.5.5 addCharacteristics

Parametri:

- `node: SelectionNode` - il nodo da aggiungere.

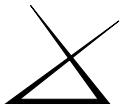
Aggiunge il nodo passato come parametro ai dati dell'albero.

5.1.5.6 removeCharacteristics

Parametri:

- `node: SelectionNode` - il nodo da rimuovere.

Rimuove il nodo passato come parametro dai dati dall'albero.



5.1.5.7 getChildrenForNode

Parametri:

- node: SelectionNode - il nodo.

Tramite un servizio che implementa `UnarchivedCharacteristicAbstractService`, tenta di ottenere la lista delle caratteristiche della macchina presente nel nodo, aggiungendole poi come figlie del nodo.

5.1.6 ChartComponent

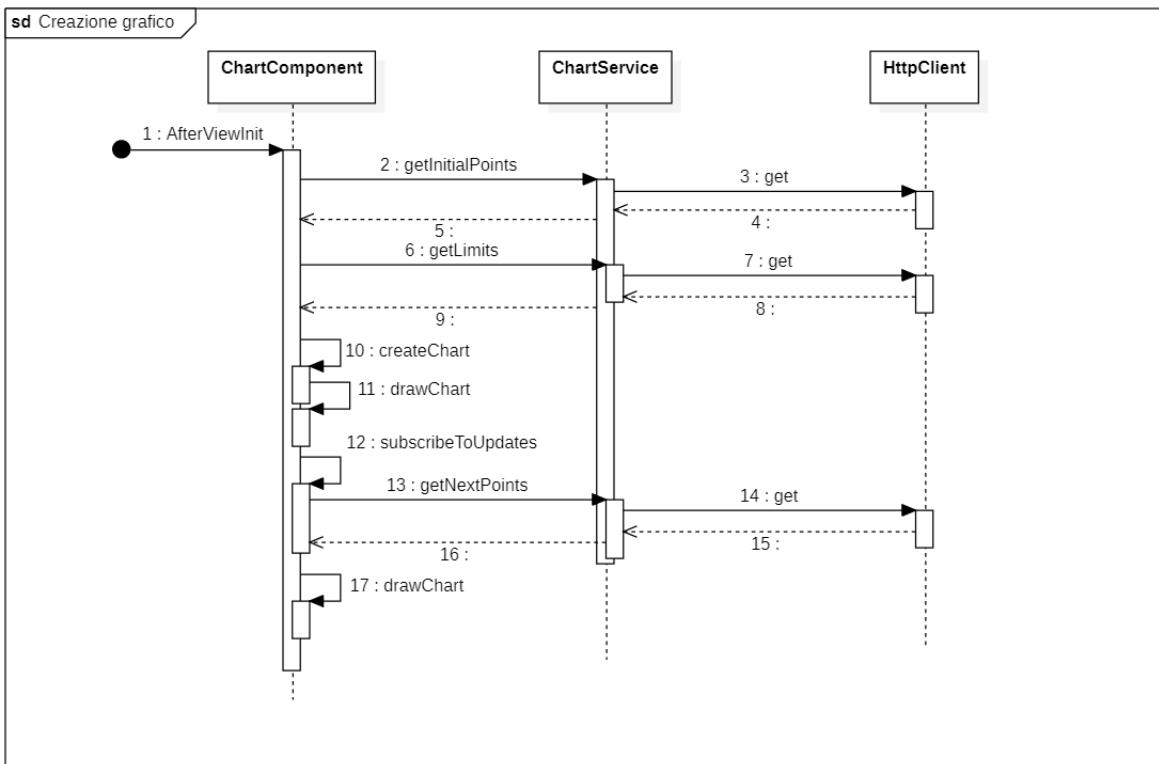


Figura 58: Diagramma di sequenza della creazione di un grafico.

Questa sequenza è innescata cliccando sul pulsante di conferma presente nell'albero di selezione caratteristiche.

1. tramite il metodo `ngAfterViewInit`, vengono chiamati prima il metodo per ottenere i limiti (`getLimits`), poi quello per ottenere i punti iniziali del grafico (`getInitialPoints`), entrambi del servizio `ChartService`;
2. entrambi i metodi effettuano delle chiamate HTTP GET al back-end, il quale risponde a entrambe con un `Observable`;
3. i valori dei limiti e dei punti vengono restituiti al contesto di `ChartComponent`;
4. vengono chiamati in sequenza i metodi `createChart` e `drawChart`;

5. viene chiamato il metodo `subscribeToUpdates` il quale, a sua volta, chiama `getNextPoints`, appartenente al servizio `ChartService`;
6. `getNextPoints` effettua una chiamata HTTP GET al back-end, ottendendo un `Observable` contenente una nuova rilevazione;
7. esso viene restituito al contesto di `ChartComponent`;
8. viene chiamato nuovamente `drawChart` per disegnare il nuovo punto.

I punti 5, 6, 7 e 8 vengono ripetuti ogni secondo.

5.1.6.1 `ngAfterViewInit`

Ereditato dall'interfaccia `AfterViewInit`. Chiama i metodi `getData()`, `createChart()` e `subscribeToUpdates()` in quest'ordine.

5.1.6.2 `ngOnDestroy`

Ereditato dall'interfaccia `OnDestroy`. Rimuove la subscription a `ChartAbstractService`.

5.1.6.3 `getChartWidth`

Ritorna la larghezza del grafico in base a quanti punti è necessario mostrare.

5.1.6.4 `getChartHeight`

Ritorna l'altezza del grafico.

5.1.6.5 `getData`

Parametri:

- `deviceId: number` - l'identificativo della macchina;
- `CharacteristicId: number` - l'identificativo della caratteristica.

Tramite un servizio che implementa `ChartAbstractService`, effettua una richiesta per ottenere i seguenti dati:

- i punti iniziali del grafico;
- i limiti superiore e inferiore;
- la media.

5.1.6.6 `createChart`

Inizializza i seguenti elementi del grafico:

- gli assi delle ascisse e delle ordinate e le relative scale;
- le linee rappresentanti limite superiore, inferiore e media;
- i punti;
- le linee che connettono i punti.

Chiama poi il metodo `drawChart`.

5.1.6.7 drawChart

Usando i dati ottenuti da `getData`, disegna gli elementi inizializzati da `createChart`.

5.1.6.8 subscribeToUpdates

Tramite un servizio che implementa `ChartAbstractService` viene effettuata una subscription che, a intervalli di un secondo, tenta di ottenere una nuova rilevazione. In caso quest'ultima venga ottenuta, viene chiamato nuovamente `drawChart` per disegnare il nuovo punto ed eliminare quello più vecchio.

5.1.6.9 clearChart

Rimuove tutti gli elementi presenti negli `svg`, svuota la variabile `points` e rimuove la subscription a `ChartAbstractService`.

5.1.6.10 refresh

Ricarica il grafico, chiamando in ordine i metodi: `clearChart`, `getData`, `createChart` e `subscribeToUpdates`.

5.1.7 DatePickerDialogComponent

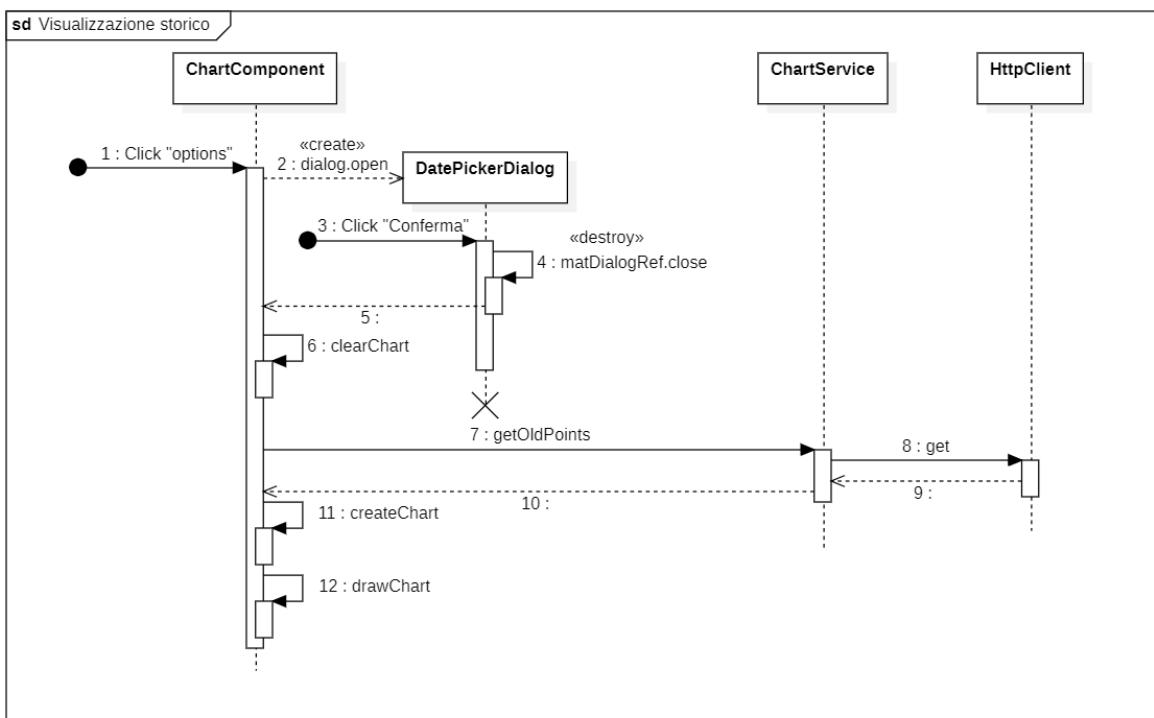
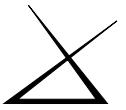


Figura 59: Diagramma di sequenza della creazione di un grafico che mostra i punti compresi tra due estremi temporali scelti dall'utente.

1. l'utente clicca sul pulsante per vedere lo storico di un grafico;



2. viene creata una finestra di dialogo in cui l'utente specifica il periodo di tempo di cui vuole vedere i punti;
3. l'utente clicca sul tasto di conferma, chiamando il metodo `confirm` che passa i dati inseriti al contesto di `ChartComponent`;
4. viene chiamato il metodo `clearChart` per rimuovere la rappresentazione precedente;
5. viene chiamato il metodo `getOldPoints` del servizio `ChartService`;
6. quest'ultimo effettua una chiamata HTTP GET al back-end, ottenendo in risposta un `Observable` contenente l'elenco delle rilevazioni richieste;
7. la risposta viene restituita al contesto di `ChartComponent`;
8. vengono chiamati, uno dopo l'altro, i metodi `createChart` e `drawChart`.

5.1.7.1 `confirm`

Prende i dati inseriti dall'utente, convertendoli in UNIX epoch. Chiude inoltre la finestra di dialogo passando i dati a `ChartComponent`, tramite il metodo `close` di `MatDialogRef`.

5.1.7.2 `cancel`

Chiude la finestra di dialogo senza alcuna operazione aggiuntiva.

5.1.8 DevicesComponent

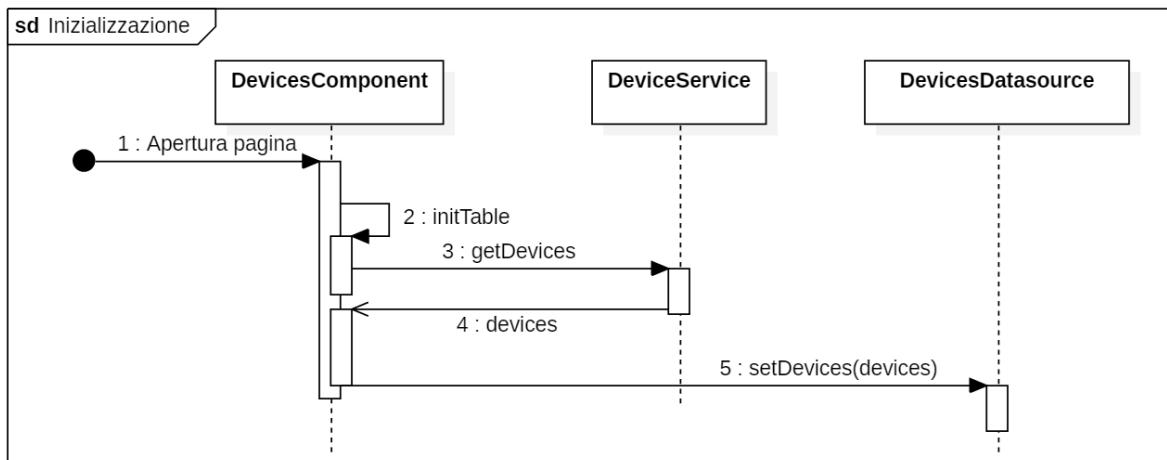
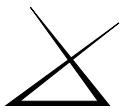


Figura 60: Diagramma di sequenza di inizializzazione componente.

1. all'apertura della pagina viene chiamato il metodo `ngOnInit`;
2. quest'ultimo chiama il metodo `initTable`;
3. quest'ultimo chiama il metodo `getDevices` del servizio dei device;
4. `getDevices` effettua una chiamata HTTP GET al back-end, il quale risponde con l'elenco delle macchine;



5. la lista delle macchine viene restituita al contesto di `initTable`, che informa la sorgente dati della tabella di mostrare le macchine ottenute.

5.1.8.1 `ngOnInit`

Ereditato dall'interfaccia `OnInit`. Inizializza i dati da visualizzare nella tabella delle macchine.

5.1.8.2 `createDevice`

Naviga alla pagina di inserimento di una nuova macchina, tramite un oggetto della classe `Router`.

5.1.8.3 `openDeviceDetail`

Parametri:

- `device: Device` - la macchina della quale vedere i dettagli.

Naviga alla pagina di dettaglio della macchina passata come parametro, tramite un oggetto della classe `Router`.

5.1.8.4 `toggleActivationDevice`

Parametri:

- `device: Device` - la macchina da attivare o disattivare.

Attiva la macchina passata come parametro se era disattivata; viceversa, la disattiva. Questo metodo si interfaccia con il server mediante un servizio che implementa `DeviceAbstractService`.

5.1.8.5 `toggleStatusDevice`

Parametri:

- `device: Device` - la macchina da archiviare o ripristinare.

Archivia la macchina passata come parametro se non lo era; viceversa, la ripristina. Si interfaccia con un servizio che implementa `DeviceAbstractService`.

5.1.8.6 `private initTable`

Inizializza la tabella con i dati provenienti dal back-end.

5.1.9 `DeviceDataSource`

5.1.9.1 `connect`

Ereditato da `DataSource`. Fornisce la sorgente dati che notifica i cambiamenti. Restituisce un `Observable<Device[]>`.

5.1.9.2 `disconnect`

Ereditato da `DataSource`. Termina il flusso di dati. Essendo una sorgente semplice, non è necessario fare nulla.

5.1.9.3 setData

Parametri:

- sort: Sort - i parametri di ordinamento.

Ordina i dati presenti nella sorgente dati.

5.1.10 NewDeviceComponent

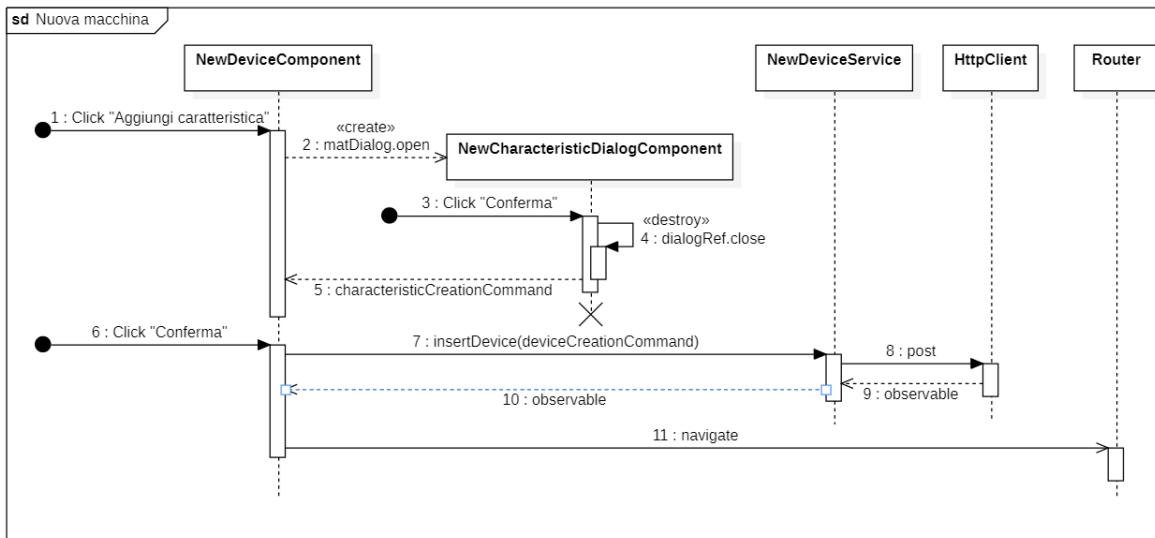


Figura 61: Diagramma di sequenza di creazione di una macchina.

L'utente accede a questa schermata cliccando sul pulsante di aggiunta macchina dalla schermata di gestione macchine.

1. l'utente clicca sul pulsante di aggiunta caratteristica;
2. crea una finestra di dialogo (mediante il metodo `open` dell'oggetto `matDialog: MatDialog`) dalla quale l'utente può inserire i dati della caratteristica;
3. l'utente inserisce i dati, poi clicca sul pulsante di conferma;
4. conseguentemente al click, la finestra di dialogo viene chiusa mediante il metodo `close` dell'oggetto `dialogRef: MatDialogRef`, notificando i dati al **NewDeviceComponent** tramite un `CharacteristicCreationCommand`;
5. il comando contiene le informazioni necessarie a creare una caratteristica. Esso viene aggiunto all'array delle caratteristiche da inserire durante la creazione della macchina;
6. l'utente inserisce altri dati della macchina. Successivamente, l'utente clicca sul pulsante di conferma dati, che chiama il servizio di creazione macchina, inserendo il `DeviceCreationCommand`;
7. il servizio effettua una chiamata HTTP POST attraverso il metodo apposito di `HttpClient`, che crea la macchina in banca dati;
8. il metodo restituisce un `Observable` al contesto del **NewDeviceComponent**;

9. al soddisfacimento della richiesta, viene effettuata una navigazione attraverso la classe Router verso la pagina di dettaglio della macchina.

5.1.10.1 ngOnInit

Ereditato da OnInit. Inizializza la logica di validazione del form.

5.1.10.2 openNewCharacteristicDialog

Apre una finestra di dialogo (classe NewCharacteristicDialogComponent) che permette di aggiungere una caratteristica alla macchina in creazione. Alla chiusura della finestra di dialogo, se l'operazione non era stata annullata, viene aggiunta la caratteristica all'elenco.

5.1.10.3 insertDevice

Tenta l'inserimento della macchina, interfacciandosi con un servizio che implementa NewDeviceAbstractService. Se l'inserimento va a buon fine, l'utente visualizza, tramite una MatSnackBar, un messaggio. Altrimenti, viene notificato al form dei dati di mostrare una spiegazione dell'errore.

5.1.10.4 private initForm

Inizializza la logica di validazione del campo del nome della macchina.

5.1.11 DeviceDetailComponent

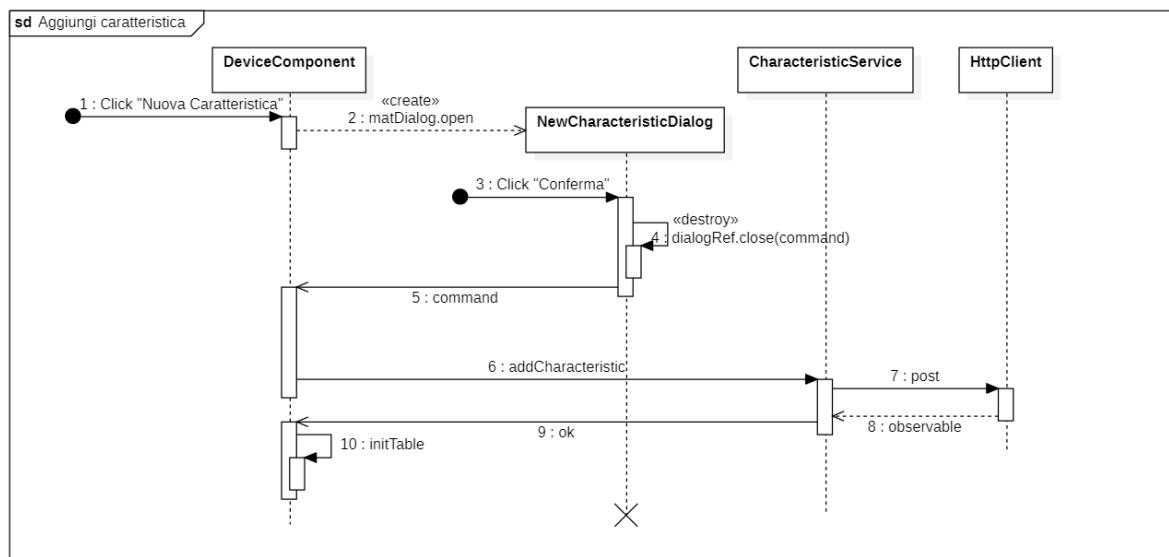


Figura 62: Diagramma di sequenza dell'aggiunta di una caratteristica a una macchina.

1. l'utente clicca sul pulsante di aggiunta caratteristica;
2. crea una finestra di dialogo (mediante il metodo `open` dell'oggetto `matDialog: MatDialog`) dalla quale l'utente può inserire i dati della caratteristica;

3. l'utente inserisce i dati, poi clicca sul pulsante di conferma. La finestra di dialogo viene chiusa mediante il metodo `close` dell'oggetto `dialogRef: MatDialogRef`, notificando i dati al `DeviceDetailComponent` tramite un `CharacteristicCreationCommand`;
4. il comando contiene le informazioni necessarie a creare una caratteristica. Esso viene fornito come parametro al metodo `insertDevice` del servizio di creazione di una caratteristica;
5. il servizio effettua una chiamata HTTP POST attraverso il metodo apposito di `HttpClient`, che crea la macchina in banca dati;
6. il metodo restituisce un `Observable` al contesto del `DeviceDetailComponent`;
7. al soddisfacimento della richiesta, vengono ricaricate le caratteristiche della macchina attraverso il metodo `initTable`.

5.1.11.1 `ngOnInit`

Ereditato da `OnInit`. Inizializza i dati da mostrare nella tabella delle caratteristiche della macchina.

5.1.11.2 `updateDeviceName`

Si interfaccia con un servizio che implementa `UpdateDeviceAbstractService` per aggiornare il nome della macchina. Ottiene le informazioni della macchina dal `FormGroup`, in caso di errore la utilizza per visualizzarlo.

5.1.11.3 `openNewCharacteristicDialog`

Apre la finestra di dialogo di aggiunta di una caratteristica, la classe `NewCharacteristicDialogComponent`, fornendo le caratteristiche attuali come dati per la validazione. Quando la finestra viene chiusa, tenta l'inserimento interfacciandosi con un servizio che implementa `CharacteristicAbstractService`, fornendogli il `CharacteristicCreationCommand` restituito dalla finestra di dialogo. Se ha successo, reinizializza la tabella delle caratteristiche.

5.1.11.4 `openUpdateCharacteristicFormDialog`

Parametri:

- `characteristic: Characteristic` - la caratteristica da modificare.

Apre la finestra di dialogo di modifica della caratteristica passata come parametro, la classe `UpdateCharacteristicDialogComponent`, inizializzata con i dati della caratteristica e l'id della macchina. Quando la finestra di dialogo viene chiusa, informa questo componente se effettuare o meno l'aggiornamento della tabella.

5.1.11.5 `notifyCopy`

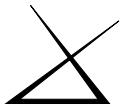
Informa l'utente tramite `MatSnackBar` che la chiave è stata copiata negli appunti (CTRL+V, o tasto destro → incolla).

5.1.11.6 `toggleCharacteristicStatus`

Parametri:

- `characteristic: Characteristic` - la caratteristica di cui cambiare lo stato di archiviazione.

Utilizza un servizio che implementa `CharacteristicAbstractService` per archiviare la caratteristica passata come parametro se non era archiviata, oppure la ripristina se era archiviata. Se la caratteristica sta per essere archiviata, viene aperta una finestra di dialogo che chiede conferma dell'operazione.



5.1.11.7 private initTable

Inizializza i dati della tabella delle caratteristiche, interfacciandosi con un servizio che implementa `CharacteristicAbstractService`.

5.1.12 DeviceDetailResolver

5.1.12.1 resolve

Parametri:

- `route: ActivatedRouteSnapshot` - il link attivato per arrivare a questa pagina.

Ereditato da `Resolve`. Ottiene la macchina indicata come parametro dalla `ActivatedRouteSnapshot` attivata. Si interfaccia con un servizio che implementa `FindDeviceAbstractService`.

5.1.13 CharacteristicDataSource

5.1.13.1 get data

Restituisce l'insieme di caratteristiche da mostrare come un array di `Characteristic`.

5.1.13.2 set data

Parametri:

- `data: Characteristic[]` - l'insieme di dati da mostrare.

Notifica il nuovo insieme di caratteristiche da mostrare nella tabella.

5.1.13.3 connect

Ereditato da `DataSource`. La sorgente dati asincrona a cui connettersi. Restituisce un `Observable<Characteristic>`.

5.1.13.4 disconnect

Ereditato da `DataSource`. Questo metodo disconnette dalla sorgente dati asincrona. Essendo una semplice sorgente dati, non occorre fare nulla.

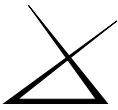
5.1.14 NewCharacteristicDialogComponent

5.1.14.1 close

Annulla l'operazione chiudendo il `MatDialogRef` senza parametri.

5.1.14.2 confirm

Notifica a chi ha aperto questa finestra di dialogo di provare a inserire la caratteristica, richiedendo i dati alla `CharacteristicFormComponent` e fornendoli al `MatDialogRef`, chiudendo la finestra di dialogo.



5.1.15 UpdateCharacteristicDialogComponent

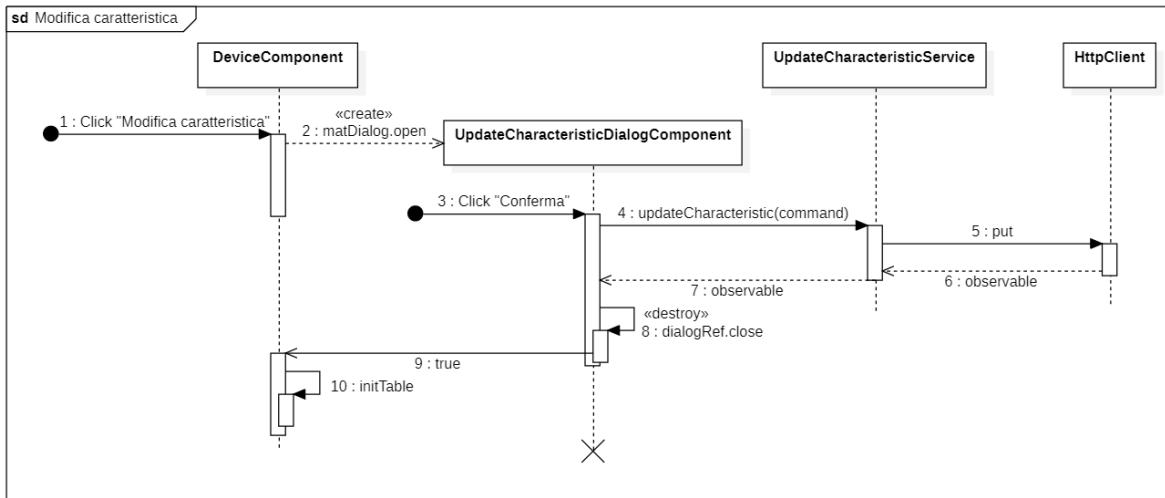


Figura 63: Diagramma di sequenza della modifica di una caratteristica.

1. l'utente clicca sul pulsante di modifica caratteristica;
2. viene creata una finestra di dialogo con il componente **UpdateCharacteristicDialogComponent**;
3. l'utente modifica la caratteristica. Successivamente, clicca il pulsante di conferma;
4. viene chiamato il metodo **updateCharateristic** del servizio di aggiornamento caratteristiche;
5. il servizio effettua una chiamata HTTP PUT mediante la classe **HttpClient**;
6. il servizio ritorna un **Observable** al contesto del **UpdateCharacteristicDialogComponent**;
7. quando l'**Observable** notifica il successo dell'operazione, la finestra di dialogo viene chiusa, passando **true** come parametro del metodo **close** del **dialogRef: MatDialogRef**;
8. il **DeviceDetailComponent** ricarica l'elenco delle caratteristiche tramite il metodo **initTable**.

5.1.15.1 close

Annulla l'operazione chiudendo il **MatDialogRef** senza parametri.

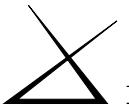
5.1.15.2 confirm

Tenta l'aggiornamento della caratteristica, richiedendo i dati alla **CharacteristicFormComponent**. L'aggiornamento avviene interfacciandosi a un servizio che implementa **UpdateCharacteristicAbstractService**. In caso di errore, viene notificato al form di mostrare il messaggio d'errore appropriato.

5.1.16 CharacteristicFormComponent

5.1.16.1 ngOnInit

Ereditato da **OnInit**. Configura la logica di validazioni dei dati del **FormGroup**. Inoltre, inizializza il form con i dati iniziali se sono presenti.



5.1.16.2 requireData

Questo metodo permette di ottenere i dati del `FormGroup` sotto forma di `CharacteristicCreationCommand`.

5.1.17 AccountsComponent

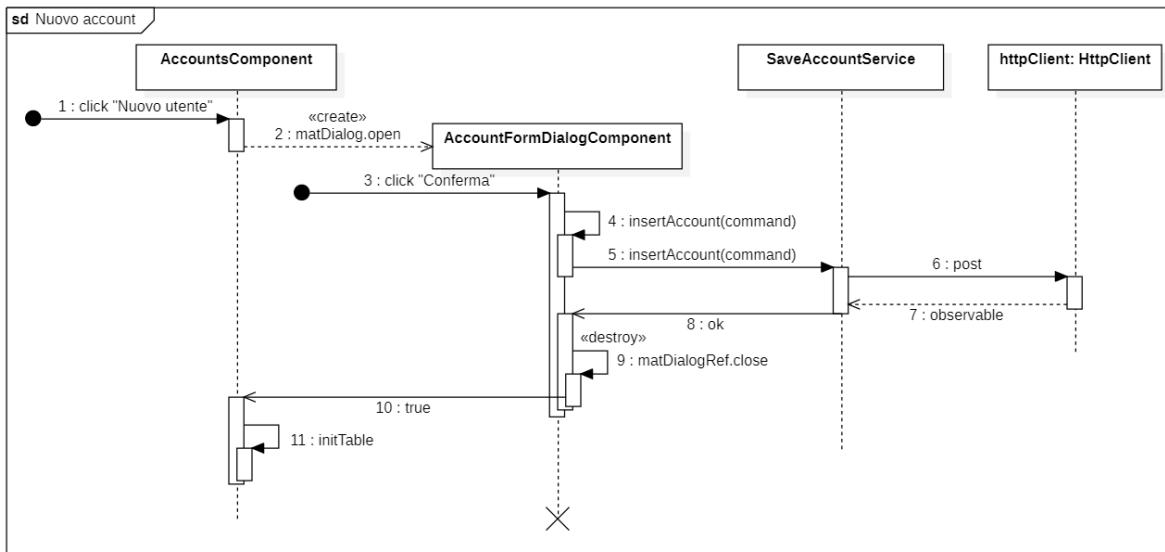
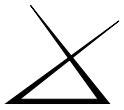


Figura 64: Diagramma di sequenza dell'inserimento di un nuovo utente.

1. l'utente clicca il pulsante di creazione nuovo utente;
2. viene creata una finestra di dialogo con il component `AccountFormDialogComponent`;
3. l'utente inserisce i dati dell'utente. Dopodiché, clicca il pulsante di conferma;
4. viene chiamata la funzione `insertAccount` con i dati inseriti come parametro, sotto forma di `AccountSaveCommand`;
5. questa funzione chiama una funzione omonima nel servizio di salvataggio degli utenti;
6. il servizio effettua una richiesta HTTP POST con i dati passati come parametro al back-end, che crea la risorsa in banca dati;
7. il `HttpClient` restituisce un `Observable`;
8. l'`Observable` restituito notifica il successo dell'operazione;
9. viene chiusa la finestra di dialogo;
10. viene informato l'`AccountsComponent` che l'operazione ha avuto successo;
11. vengono ricaricati i dati da mostrare in interfaccia tramite la funzione `initTable`.

5.1.17.1 ngOnInit

Ereditato da `OnInit`. Viene usato per inizializzare la tabella.



5.1.17.2 openNewAccountDialog

Apre una finestra di dialogo `AccountFormDialog` in modalità di creazione. Alla chiusura della finestra, se l'operazione non era stata annullata e ha avuto successo, vengono ricaricati gli utenti da mostrare nella tabella.

5.1.17.3 openEditAccountDialog

Parametri:

- `account`: `Account` - l'utente da modificare.

Apre una finestra di dialogo `AccountFormDialog` in modalità di modifica, inizializzata con i dati dell'utente, passati come parametro. Alla chiusura della finestra, se l'operazione non era stata annullata e ha avuto successo, vengono ricaricati gli utenti da mostrare nella tabella.

5.1.17.4 toggleAccountStatus

Parametri:

- `account`: `Account` - l'utente da modificare.

Archivia l'account passato come parametro se non era archiviato, altrimenti lo ripristina. Dopodiché notifica l'esito all'utente tramite una `MatSnackBar`.

5.1.17.5 isLoggedInUser

Parametri:

- `account`: `Account` - l'utente da verificare se è registrato o meno.

Restituisce `true` solo se l'utente passato come parametro è l'utente registrato.

5.1.17.6 private initTable

Imposta gli utenti da visualizzare nella tabella, interfacciandosi con un servizio che implementa `AccountAbstractService`.

5.1.18 AccountsDataSource

5.1.18.1 connect

Ereditato da `DataSource`. Restituisce la sorgente dati a cui agganciarsi.

5.1.18.2 disconnect

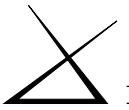
Ereditato da `DataSource`. Non esegue nulla.

5.1.18.3 setData

Parametri:

- `data`: `Account[]` - gli utenti da visualizzare.

Notifica alla sorgente dati di visualizzare gli utenti passati come parametro.



5.1.18.4 sortData

Parametri:

- sort: Sort - le indicazioni su come ordinare i dati.

Ordina i dati visualizzati.

5.1.19 AccountFormDialogComponent

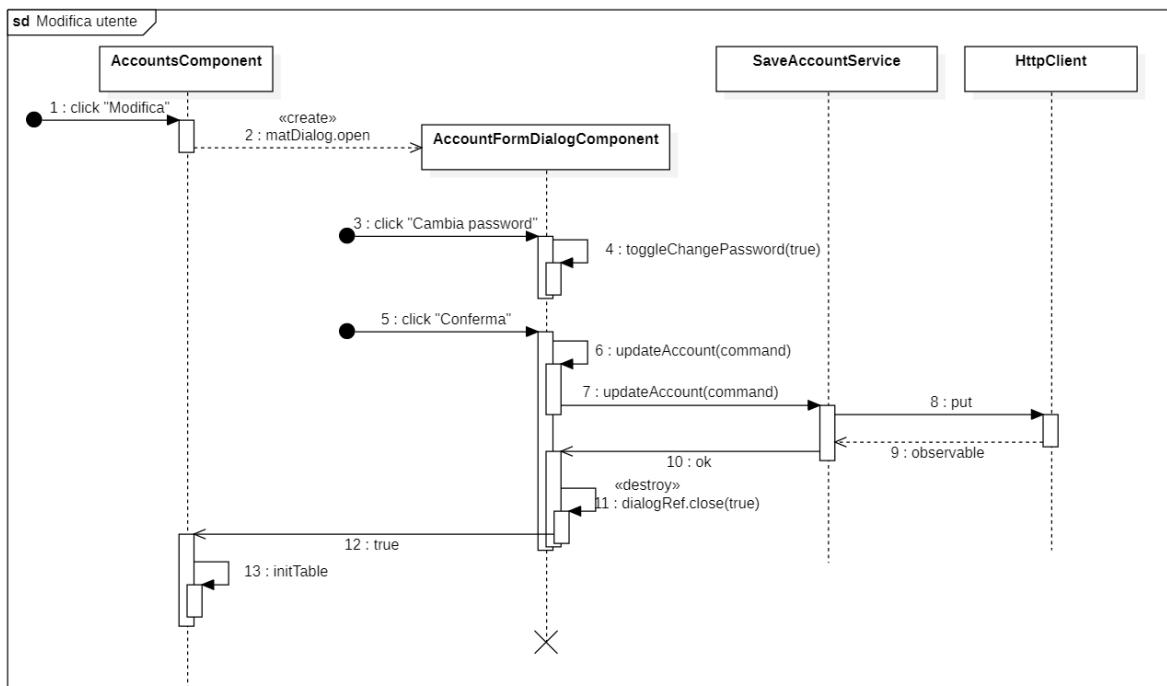


Figura 65: Diagramma di sequenza della modifica di un utente.

- l'amministratore clicca sul pulsante di modifica utente;
- viene aperta una finestra di dialogo **AccountFormDialogComponent** inizializzata con le informazioni di quell'utente;
- l'amministratore modifica i dati dell'utente. Tra questi dati, abilita il cambiamento della password cliccando l'apposito pulsante;
- viene chiamata la funzione `toggleChangePassword` con parametro `true`. L'amministratore può quindi modificare la password dell'utente;
- l'amministratore clicca il bottone di conferma;
- viene chiamata la funzione `updateAccount` di **AccountFormDialogComponent**;
- viene chiamata la funzione `updateAccount` del servizio di salvataggio degli utenti;
- il servizio effettua una richiesta HTTP PUT al back-end con i parametri forniti, modificando la risorsa;

- il servizio restituisce un `Observable`;
- l'`Observable` notifica che l'operazione è andata a buon fine;
- la finestra di dialogo viene chiusa;
- l'`AccountsComponent` viene notificato del successo dell'operazione;
- l'`AccountsComponent` ricarica i dati da mostrare mediante `initTable`.

5.1.19.1 `ngOnInit`

Ereditato da `OnInit`. Viene usato per configurare la logica di validazione del form.

5.1.19.2 `cancel`

Annulla l'operazione, chiudendo la finestra di dialogo tramite `MatDialogRef` senza parametri.

5.1.19.3 `confirm`

Tenta la conferma dell'operazione, prelevando i dati dal `FormGroup`. In caso di successo, chiude la finestra di dialogo tramite il `MatDialogRef`, con `true` come parametro; altrimenti visualizza l'errore.

5.1.19.4 `toggleChangePassword`

Parametri:

- `enabled`: `boolean` - rappresenta lo stato da impostare alla form.

Se il parametro è `true`, abilita il form per la modifica della password; altrimenti lo imposta per mantenere la password attuale.

5.1.19.5 `private insertAccount`

Parametri:

- `command`: `AccountSaveCommand` - le informazioni del nuovo utente.

Tenta l'inserimento di un nuovo utente tramite un `AccountSaveCommand`, interfacciandosi con un servizio che implementa `SaveAccountAbstractService`. Restituisce un `Observable` dell'operazione.

5.1.19.6 `private updateAccount`

Parametri:

- `command`: `AccountSaveCommand` - le informazioni di modifica dell'utente.

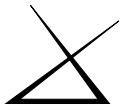
Tenta la modifica di un utente esistente, tramite un `AccountSaveCommand`, interfacciandosi con un servizio che implementa `SaveAccountAbstractService`. Restituisce un `Observable` dell'operazione.

5.1.19.7 `private showError`

Parametri:

- `err`: `StandardError` - l'errore da gestire.

Incapsula la logica di visualizzazione dell'errore. Se l'errore è conosciuto, lo imposta come errore del form, altrimenti lo visualizza in un `ErrorDialogContent`.



5.2 Back-end UI

5.2.1 AccountsController

5.2.1.1 updateAccountPassword

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/username/password` con metodo PUT. Inoltre quindi l'aggiornamento appena ricevuto da effettuare al metodo `updatePasswordByUsername` del campo `updateAccountPasswordUseCase`. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200.

5.2.2 AdminsAccountsController

5.2.2.1 insertAccount

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/admin/accounts` con metodo POST. Inoltre quindi l'utente appena ricevuto da memorizzare al metodo `insertAccount` del campo `insertAccountUseCase`. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200 e una mappa con l'username dell'utente appena aggiunto.

5.2.2.2 getAccounts

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/admin/accounts` con metodo GET. Inoltre quindi la richiesta al metodo `getTinyAccounts` del campo `getAccountsUseCase`. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200 e la lista degli utenti.

5.2.2.3 updateAccount

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/admin/username` con metodo PUT. Inoltre quindi la richiesta al metodo `updateByUsername` del campo `updateAccountByAdminUseCase`. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200.

5.2.2.4 updateAccountArchiveStatus

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/admin/accounts/username/archived` con metodo PUT. Inoltre quindi la richiesta al metodo `updateAccountArchiveStatus` del campo `updateAccountArchiveStatusUseCase`. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200.

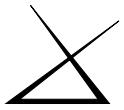
5.2.3 AdminsCharacteristicsController

5.2.3.1 insertCharacteristic

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/admin/devices/{deviceId}/characteristics` con metodo POST. Inoltre quindi l'inserimento della caratteristica appena ricevuta al metodo `insertByDevice` del campo `insertCharacteristicUseCase`, passandogli l'identificativo della macchina e i dettagli della caratteristica in un'istanza del record `NewCharacteristic`. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200 e l'identificativo della caratteristica, relativo alla macchina.

5.2.3.2 getCharacteristics

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/admins/devices/{deviceId}/characteristics` con metodo GET. Inoltre quindi la richiesta dell'ottenimento della lista delle caratteristiche al metodo `getByDevice` del campo `getCharacteristicsUseCase`, passandogli l'identificativo della macchina. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200 e la lista.



5.2.3.3 updateCharacteristicArchiveStatus

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/admins/devices/{deviceId}/characteristics/{characteristicId}/archived` con metodo PUT. Inoltre quindi la richiesta della modifica dello stato di archiviazione della caratteristica al metodo `updateCharacteristicArchiveStatus` del campo `updateCharacteristicArchiveStatusUseCase`, passandogli i dettagli della richiesta in un'istanza del record `CharacteristicArchiveStatus`. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200.

5.2.3.4 updateCharacteristic

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/admins/devices/{deviceId}/characteristics/{characteristicId}` con metodo PUT. Inoltre quindi la richiesta della modifica della caratteristica al metodo `updateCharacteristic` del campo `updateCharacteristicUseCase`, passandogli i dettagli della richiesta in un'istanza del record `CharacteristicToUpdate`. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200.

5.2.4 AdminsDevicesController

5.2.4.1 insertDevice

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/admin/devices` con metodo POST. Inoltre quindi l'inserimento della macchina appena ricevuta al metodo `insertDevice` del campo `insertDeviceUseCase`. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200 e l'identificativo della macchina.

5.2.4.2 getDevices

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/admin/devices` con metodo GET. Inoltre quindi la richiesta dell'ottenimento della lista delle macchine al metodo `getDevices` del campo `getDevicesUseCase`. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200 e la lista.

5.2.4.3 getDeviceDetails

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/admin/devices/{id}` con metodo GET. Inoltre quindi la richiesta dell'ottenimento di dettagli della macchina al metodo `getDeviceDetails` del campo `getDeviceDetailsUseCase`, passandogli l'identificativo della macchina. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200 e la macchina con i dettagli.

5.2.4.4 updateDeviceName

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/admin/devices/{id}` con metodo PUT. Inoltre quindi la richiesta dell'aggiornamento del nome della macchina al metodo `updateDeviceName` del campo `updateDeviceNameUseCase`, passandogli l'identificativo della macchina e il nome. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200.

5.2.4.5 updateDeviceArchiveStatus

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/admin/devices/{id}/archived` con metodo PUT. Inoltre quindi la richiesta dell'aggiornamento dello stato di archiviazione della macchina al metodo `updateDeviceArchiveStatus` del campo `updateDeviceArchiveStatusUseCase`, passandogli l'identificativo della macchina e lo stato di archiviazione da impostare. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200.

5.2.4.6 updateDeviceDeactivateStatus

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/admin/devices/{id}/deactivated` con metodo PUT. Inoltra quindi la richiesta dell'aggiornamento dello stato di attivazione della macchina al metodo `updateDeviceDeactivateStatus` del campo `updateDeviceDeactivateStatusUseCase`, passandogli l'identificativo della macchina e lo stato di attivazione da impostare. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200.

5.2.5 CharacteristicsController

5.2.5.1 getUnarchivedCharacteristics

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/devices/{deviceId}/characteristics` con metodo GET. Inoltra quindi la richiesta dell'ottenimento della lista delle caratteristiche non archiviate al metodo `getByDevice` del campo `getUnarchivedCharacteristicsUseCase`, passandogli l'identificativo della macchina. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200 e la lista.

5.2.5.2 getCharacteristicLimits

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/devices/{deviceId}/characteristics/{characteristicId}/limits` con metodo GET. Inoltra quindi la richiesta dell'ottenimento dei limiti della caratteristica al metodo `getByCharacteristic` del campo `getLimitsUseCase`, passandogli l'identificativo della macchina e quello della caratteristica, relativo alla macchina. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200 e i limiti della caratteristica.

5.2.6 DetectionsController

5.2.6.1 getCharacteristicDetections

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/devices/{deviceId}/characteristics/{characteristicId}/detections` con metodo GET. Inoltra quindi la richiesta dell'ottenimento della lista delle rilevazioni della caratteristica al metodo `listByCharacteristic` del campo `getDetectionsUseCase`, passandogli l'identificativo della macchina e quello della caratteristica, relativo alla macchina e i filtri di ricerca richiesti. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200 e la lista.

5.2.7 DevicesController

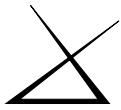
5.2.7.1 getUnarchivedDevices

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/devices` con metodo GET. Inoltra quindi la richiesta dell'ottenimento della lista delle macchine non archiviate al metodo `getUnarchivedDevices` del campo `getUnarchivedDevicesUseCase`. Se tutto è andato a buon fine, ritorna il codice di stato HTTP 200 e la lista.

5.2.8 LoginController

5.2.8.1 login

Questo metodo si occupa di ricevere le chiamate effettuate all'endpoint `/login` con metodo GET. Il corpo del metodo è vuoto, la logica è gestita da Spring Security.



5.2.9 BusinessExceptionHandler

5.2.9.1 handleStatusException

Questo metodo si occupa di produrre una risposta di errore in base all'eccezione lanciata. Deve quindi:

- individuare il codice di stato HTTP, ovvero un oggetto di tipo `HttpStatus` in base al tipo di errore, che può ottenere tramite `BusinessException::getType`, in particolare:
 - `ErrorType.GENERIC` corrisponde al codice 400, ovvero `HttpStatus.GENERIC`;
 - `ErrorType.AUTHENTICATION` corrisponde al codice 401, ovvero `HttpStatus.UNAUTHORIZED`;
 - `ErrorType.NOT_FOUND` corrisponde al codice 404, ovvero `HttpStatus.NOT_FOUND`.
- creare il corpo della risposta, ovvero creare una mappa che associa la stringa "errorCode" al codice di errore ottenuto tramite `BusinessException::getCode`;
- creare l'oggetto di risposta, ovvero un `ResponseEntity`, con i due valori appena individuati.

5.2.10 BusinessException

5.2.10.1 getCode

Questo metodo deve ritornare il valore del campo `code`.

5.2.10.2 getType

Questo metodo deve ritornare il valore del campo `type`.

5.2.11 AccountAdapter

5.2.11.1 findByUsername

Questo metodo invoca il metodo `findById` del campo `repo` per cercare l'utente. Infine, restituisce il risultato nel formato `Account`.

5.2.11.2 updateAccountPassword

Questo metodo invoca il metodo `save` del campo `repo` per aggiornare la password dell'utente.

5.2.12 AdminAccountAdapter

5.2.12.1 updateAccountArchiveStatus

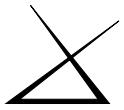
Questo metodo invoca il metodo `save` del campo `repo` per aggiornare l'utente con lo stato di archiviazione modificato.

5.2.12.2 findByUsername

Questo metodo invoca il metodo `findById` del campo `repo` per cercare l'utente. Infine, restituisce il risultato nel formato `Account`.

5.2.12.3 updateAccount

Questo metodo invoca il metodo `save` del campo `repo` per aggiornare l'utente con le modifiche che sono state effettuate.



5.2.12.4 insertAccount

Questo metodo invoca il metodo `save` del campo `repo` per memorizzare l'utente `account`.

5.2.12.5 getTinyAccounts

Questo metodo invoca il metodo `findAll` del campo `repo` per ottenere lo `stream` con gli utenti esistenti. Infine, restituisce il risultato in una lista di elementi nel formato `TinyAccount`.

5.2.13 AdminDeviceAdapter

5.2.13.1 getDevices

Questo metodo invoca il metodo `findAll` del campo `repo` per ottenere lo `stream` delle macchine esistenti. Infine, restituisce il risultato in una lista di elementi nel formato `Device`.

5.2.13.2 findTinyDevice

Questo metodo invoca il metodo `findById` del campo `repo` per cercare una macchina. Infine, restituisce il risultato in una lista di elementi nel formato `TinyDevice`.

5.2.13.3 findDetailedDevice

Questo metodo invoca il metodo `findById` del campo `repo` per cercare una macchina. Infine, restituisce il risultato in una lista di elementi nel formato `DetailedDevice`.

5.2.13.4 updateDeviceArchiveStatus

Questo metodo invoca il metodo `save` del campo `repo` per memorizzare lo stato di archiviazione modificato della macchina.

5.2.13.5 updateDeviceDeactivateStatus

Questo metodo invoca il metodo `save` del campo `repo` per memorizzare lo stato di attivazione modificato della macchina.

5.2.13.6 updateDeviceName

Questo metodo invoca il metodo `save` del campo `repo` per memorizzare il nome modificato della macchina.

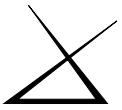
5.2.13.7 insertDevice

Questo metodo invoca il metodo `saveAndFlush` del campo `repo` per memorizzare la nuova macchina e averla subito disponibile. Infine, restituisce l'identificativo della macchina appena inserita.

5.2.14 AdminCharacteristicAdapter

5.2.14.1 findAllByDeviceId

Questo metodo invoca il metodo `findById` del campo `repo`, passandogli l'identificativo della macchina, per ottenere la lista delle caratteristiche. Infine, restituisce il risultato in una lista di elementi nel formato `Characteristic`.



5.2.14.2 `findByCharacteristic`

Questo metodo invoca il metodo `findById` del campo `repo` e restituisce la caratteristica ottenuta, utilizzando il metodo `toDetailed` dell'interfaccia `ConvertCharacteristic`.

5.2.14.3 `findByDeviceAndName`

Questo metodo invoca il metodo `findByIdAndName` del campo `repo` e restituisce la lista ottenuta, con le caratteristiche utilizzando il metodo `toDetailed` dell'interfaccia `ConvertCharacteristic`.

5.2.14.4 `insertByDevice`

Questo metodo memorizza la caratteristica passata nella macchina con l'identificativo passato invocando il metodo `save` del campo `repo` e restituisce l'identificativo della caratteristica, relativo alla macchina, inserita.

5.2.14.5 `updateCharacteristic`

Questo metodo memorizza le nuove informazioni della caratteristica passata invocando il metodo `save` del campo `repo`.

5.2.15 `DetectionAdapter`

5.2.15.1 `findAllByCharacteristic`

Questo metodo invoca il metodo `findByCharacteristicAndCreationTimeGreaterThanOrQuery` del campo `repo`, passandogli l'identificativo della macchina e quello della caratteristica, relativo alla macchina e il filtro opzionale di ricerca `olderThan`, per ottenere la lista delle rilevazioni della caratteristica, ordinandole in ordine decrescente rispetto all'istante di creazione. Infine, restituisce il risultato in una lista di elementi nel formato `Detection`.

5.2.16 `PasswordEncoderAdapter`

5.2.16.1 `encode`

Questo metodo invoca il metodo `encode` del campo `encoderConfig`, per cifrare la password in chiaro. Infine, restituisce la password cifrata.

5.2.17 `PasswordMatcherAdapter`

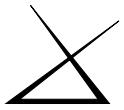
5.2.17.1 `matches`

Questo metodo invoca il metodo `matches` su del campo `encoderConfig`, per confrontare la password in chiaro con quella cifrata. Infine, restituisce l'esito della comparazione.

5.2.18 `UnarchivedCharacteristicAdapter`

5.2.18.1 `findAllByDeviceId`

Questo metodo invoca il metodo `findByArchivedFalseAndDeviceId` del campo `repo`, passandogli l'identificativo della macchina, per ottenere la lista delle caratteristiche non archiviate. Infine, restituisce il risultato in una lista di elementi nel formato `TinyCharacteristic`.



5.2.18.2 `findByCharacteristic`

Questo metodo invoca il metodo `findByArchivedFalseAndDeviceIdAndId` del campo `repo`, passandogli l'identificativo della macchina e quello della caratteristica, relativo alla macchina, per ottenere i limiti della caratteristica. Infine, restituisce il risultato in una lista di elementi nel formato `CharacteristicLimits`.

5.2.19 `UserDetailsAdapter`

5.2.19.1 `loadUserByUsername`

Questo metodo:

- controlla che esista un utente con l'username dato tramite l'invocazione del metodo `findByUsername` del campo `findAccountUseCase`;
- se esiste, restituisce una nuova istanza di `CustomAccountDetails`;
- altrimenti, lancia un'eccezione di tipo `UsernameNotFoundException` con messaggio "usernameNotFound".

5.2.20 `DetectionRepository`

5.2.20.1 `findByCharacteristicAndCreationTimeGreaterThanQuery`

Questo metodo è implementato da Spring secondo la query passata all'annotazione `@Query`. Essa deve selezionare tutte le rilevazioni che hanno gli identificativi della macchina e della caratteristica passati per parametro e che sono state create prima dell'istante passato. Se non viene specificato l'istante, cioè è pari a `null`, vengono restituite tutte le rilevazioni associate alla macchina e alla caratteristica specificate. Il risultato restituito viene ordinato secondo l'ordine passato per parametro.

5.2.21 `FindAccountService`

5.2.21.1 `findByUsername`

Questo metodo:

- mediante il metodo `findByUsername` del campo `findAccountPort` verifica che l'utente esista, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "accountNotFound" e codice `ErrorType.NOT_FOUND`;
- se l'utente è stato trovato, lo restituisce.

5.2.22 `GetTinyAccountsService`

5.2.22.1 `getTinyAccounts`

Questo metodo restituisce il risultato dell'invocazione del metodo `getTinyAccounts` del campo `getTinyAccountsPort`.

5.2.23 `InsertAccountService`

5.2.23.1 `insertAccount`

Questo metodo:

- controlla che la password inserita rispetti i requisiti di lunghezza, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "invalidPassword" e codice `ErrorType.GENERIC`;

- mediante il metodo `findByUsername` del campo `findAccountByAdminPort`, verifica se esiste già un utente con lo stesso username, se è così lancia un'eccezione di tipo `BusinessException` con messaggio "duplicateUsername" e codice `ErrorType.GENERIC`;
- se tutti i controlli sono andati a buon fine, passa al metodo `insertAccount` del campo `insertAccountPort` l'utente da memorizzare.

5.2.24 UpdateAccountArchiveStatusService

5.2.24.1 updateAccountArchiveStatus

Questo metodo:

- mediante il metodo `findByUsername` del campo `findAccountByAdminPort`, verifica che l'utente esista. Se non è così, lancia un'eccezione di tipo `BusinessException` con messaggio "accountNotFound" e codice `ErrorType.NOT_FOUND`;
- se l'utente esiste, passa al metodo `updateAccountArchiveStatus` del campo `updateAccountArchiveStatusPort` l'utente con il nuovo stato di archiviazione da memorizzare.

5.2.25 UpdateAccountByAdminService

5.2.25.1 updateByUsername

Questo metodo:

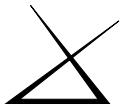
- se è presente la nuova password, controlla che rispetti i requisiti di lunghezza; se non li rispetta lancia un'eccezione di tipo `BusinessException` con messaggio "invalidNewPassword" e codice `ErrorType.GENERIC`;
- mediante il metodo `findByUsername` del campo `findAccountByAdminPort` verifica che l'utente esista, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "accountNotFound" e codice `ErrorType.NOT_FOUND`;
- se è presente la nuova password, invoca il metodo `encode` del campo `passwordEncoderPort` e memorizza in `hashedPassword` la password cifrata restituita;
- se tutti i controlli sono andati a buon fine, invoca il metodo `updateAccount` del campo `updateAccountByAdminPort`.

5.2.26 UpdateAccountPasswordService

5.2.26.1 updatePasswordByUsername

Questo metodo:

- controlla che la nuova password rispetti i requisiti di lunghezza, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "invalidNewPassword" e codice `ErrorType.GENERIC`;
- mediante il metodo `findByUsername` del campo `findAccountByAdminPort` verifica che l'utente esista, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "accountNotFound" e codice `ErrorType.NOT_FOUND`;
- mediante il metodo `matches` del campo `passwordMatcherPort` verifica che la password corrente inserita sia corretta, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "wrongCurrentPassword" e codice `ErrorType.AUTHENTICATION`;
- se tutti i controlli sono andati a buon fine, passa al metodo `updateAccountPassword` del campo `updateAccountPasswordPort` l'utente modificato da memorizzare.



5.2.27 CreateDevice

5.2.27.1 generateApiKey

Questo metodo restituisce una stringa casuale lunga 32 caratteri alfanumerici.

5.2.27.2 createDevice

Questo metodo restituisce una nuova istanza di `NewDevice` con nome uguale al parametro `name`, stato di attivazione e archiviazione falsi e con chiave per l'API rilevazioni generata tramite l'invocazione del metodo `generateApiKey`.

5.2.28 GetDeviceDetailsService

5.2.28.1 getDeviceDetails

Questo metodo:

- mediante il metodo `findDetailedDevice` del campo `findDetailedDevicePort` ottiene i dettagli sulla macchina, li converte nel formato `DetailedDevice` e li restituisce;
- se la macchina chiamata non restituisce la macchina, allora lancia un'eccezione di tipo `BusinessException` con messaggio "deviceNotFound" e codice `ErrorType.NOT_FOUND`.

5.2.29 GetDevicesService

5.2.29.1 getDevices

Questo metodo invoca il metodo `getDevices` del campo `getDevicesPort` e ne restituisce il risultato.

5.2.30 GetUnarchivedDevicesService

5.2.30.1 getUnarchivedDevices

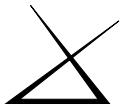
Questo metodo inoltra la richiesta invocando il metodo `texttt{getUnarchivedDevices}` del campo `getUnarchivedDevicesPort` e ne ritorna il risultato.

5.2.31 InsertDeviceService

5.2.31.1 insertDevice

Questo metodo:

- mediante il metodo `createDevice` del campo `createDevicePort` costruisce una nuova macchina con nome uguale al parametro `name`;
- mediante il metodo `insertDevice` del campo `insertDevicePort` memorizza la macchina creata;
- per ogni caratteristica da memorizzare nella macchina, invoca il metodo `insertByDevice` del campo `insertCharacteristicUseCase`;
- infine, restituisce l'identificativo della macchina appena creata.



5.2.32 UpdateDeviceArchiveStatusService

5.2.32.1 updateDeviceArchiveStatus

Questo metodo:

- mediante il metodo `findDetailedDevice` del campo `findDetailedDevicePort` verifica che la macchina esista, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "deviceNotFound" e codice `ErrorType.NOT_FOUND`;
- mediante il metodo `updateDeviceArchiveStatus` del campo `updateDeviceArchiveStatus` memorizza la macchina con lo stato di archiviazione modificato.

5.2.33 UpdateDeviceDeactivateStatusService

5.2.33.1 updateDeviceDeactivateStatus

Questo metodo:

- mediante il metodo `findDetailedDevice` del campo `findDetailedDevicePort` verifica che la macchina esista, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "deviceNotFound" e codice `ErrorType.NOT_FOUND`;
- mediante il metodo `updateDeviceDeactivateStatus` del campo `updateDeviceDeactivateStatusPort` memorizza la macchina con lo stato di attivazione modificato.

5.2.34 UpdateDeviceNameService

5.2.34.1 updateDeviceName

Questo metodo:

- mediante il metodo `findDetailedDevice` del campo `findDetailedDevicePort` verifica che la macchina esista, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "deviceNotFound" e codice `ErrorType.NOT_FOUND`;
- mediante il metodo `updateDeviceName` del campo `updateDeviceNamePort` memorizza la macchina con il nome aggiornato.

5.2.35 CharacteristicConstraints

5.2.35.1 characteristicConstraintsOk

Questo metodo restituisce `true` se i vincoli di integrità della caratteristica passata sono validi; `false` altrimenti.

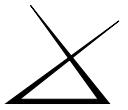
5.2.36 ConvertCharacteristic

5.2.36.1 toDetailed

Questo metodo converte un'istanza del record `CharacteristicEntity` passato in una equivalente del record `DetailedCharacteristic`.

5.2.36.2 toEntity

Questo metodo converte un'istanza del record `DetailedCharacteristic` passato in una equivalente del record `CharacteristicEntity`.



5.2.37 GetCharacteristicsService

5.2.37.1 getByDevice

Questo metodo inoltra la richiesta invocando il metodo `findAllByIdDevice` del campo `findCharacteristicsPort`, passandogli l'identificativo della macchina, e ne ritorna il risultato se la lista restituita contiene almeno una caratteristica, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "deviceNotFound" e codice `ErrorType.NOT_FOUND`.

5.2.38 InsertCharacteristicService

5.2.38.1 insertByDevice

Questo metodo:

1. mediante il metodo `findDetailedDevice` del campo `findDevicePort` verifica che la macchina con l'identificativo passato esista, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "deviceNotFound" e codice `ErrorType.NOT_FOUND`;
2. mediante il metodo `findByDeviceAndName` del campo `findCharacteristicPort` verifica che la caratteristica da inserire non abbia il nome di una già esistente, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "duplicateCharacteristicName" e codice `ErrorType.GENERIC`;
3. mediante il metodo `characteristicConstraintsOk` dell'interfaccia `CharacteristicConstraints` verifica che i vincoli di integrità della caratteristica da inserire siano validi, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "invalidValues" e codice `ErrorType.GENERIC`;
4. procede con l'inserimento della caratteristica, invocando il metodo `insertByDevice` del campo `insertCharacteristicPort`;
5. infine, restituisce l'identificativo della caratteristica, relativo alla macchina, appena inserita.

5.2.39 UpdateCharacteristicArchiveStatusService

Questo metodo:

- mediante il metodo `findByCharacteristic` del campo `findCharacteristicPort` verifica che la caratteristica esista e ne ottiene i dettagli precedenti alla modifica, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "characteristicNotFound" e codice `ErrorType.NOT_FOUND`;
- mediante il metodo `updateCharacteristic` del campo `updateCharacteristicPort` memorizza le informazioni della caratteristica ottenute precedentemente, modificandone solamente lo stato di archiviazione.

5.2.40 UpdateCharacteristicService

Questo metodo:

1. mediante il metodo `findByCharacteristic` del campo `findCharacteristicPort` verifica che la caratteristica esista e ne ottiene il valore del campo `archived` precedente alla modifica, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "characteristicNotFound" e codice `ErrorType.NOT_FOUND`;

2. mediante il metodo `findByDeviceAndName` del campo `findCharacteristicPort` verifica che non esista già una caratteristica con il nuovo nome desiderato, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "duplicateCharacteristicName" e codice `ErrorType.GENERIC`;
3. mediante il metodo `characteristicConstraintsOk` dell'interfaccia `CharacteristicConstraints` verifica che i vincoli di integrità della caratteristica da modificare siano validi, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "invalidValues" e codice `ErrorType.GENERIC`;
4. infine, procede con la modifica della caratteristica, invocando il metodo `updateCharacteristic` del campo `updateCharacteristicPort`.

5.2.41 GetUnarchivedCharacteristicsService

5.2.41.1 getByDevice

Questo metodo:

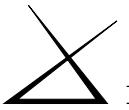
- cerca la macchina con l'identificativo passato invocando il metodo `getUnarchivedDevices` del campo `findDevicesPort` per verificare che esista, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "deviceNotFound" e codice `ErrorType.NOT_FOUND`;
- inoltra la richiesta invocando il metodo `findAllByDeviceId` del campo `findCharacteristicsPort`, passandogli l'identificativo della macchina, e ne ritorna il risultato.

5.2.42 GetDetectionsService

5.2.42.1 listByCharacteristic

Questo metodo:

- mediante il metodo `findByCharacteristic` del campo `findCharacteristicPort` verifica che la caratteristica esista, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "characteristicNotFound" e codice `ErrorType.NOT_FOUND`;
- ottiene la lista delle rilevazioni, ordinate in ordine decrescente rispetto all'istante di creazione, invocando il metodo `findAllByCharacteristic` del campo `findDetectionsPort`, passandogli l'identificativo della macchina e quello della caratteristica, relativo alla macchina e il filtro opzionale di ricerca `olderThan`, e ne memorizza la lunghezza;
- se il filtro opzionale di ricerca `limit` è presente, viene tenuto, al massimo, il numero di rilevazioni più recenti specificato dal filtro;
- se il filtro opzionale di ricerca `newerThan` è presente, vengono tenute solamente le rilevazioni che hanno un istante di creazione successivo a quello specificato dal filtro;
- calcola i valori `nextNew` e `nextOld`;
- restituisce una nuova istanza del record `DetectionsGroup`, contenente la lista di rilevazioni elaborata, in ordine inverso, e i valori di `nextNew` e `nextOld`.



5.2.43 GetLimitsService

5.2.43.1 getByCharacteristic

Questo metodo:

- cerca la macchina con l'identificativo passato invocando il metodo `getUnarchivedDevices` del campo `findDevicesPort` per verificare che esista, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "`characteristicNotFound`" e codice `ErrorType.NOT_FOUND`;
- inoltra la richiesta invocando il metodo `findByCharacteristic` del campo `findCharacteristicLimitsPort`, passandogli l'identificativo della macchina e quello della caratteristica, relativo alla macchina, e ne ritorna il risultato se la caratteristica esiste, se non è così lancia un'eccezione di tipo `BusinessException` con messaggio "`characteristicNotFound`" e codice `ErrorType.NOT_FOUND`.

5.2.44 CustomAccountDetails

5.2.44.1 getUsername

Questo metodo invoca il metodo `getUsername` del campo `account` e restituisce la stringa ottenuta.

5.2.44.2 getPassword

Questo metodo invoca il metodo `getPassword` del campo `account` e restituisce la stringa ottenuta.

5.2.44.3 getAuthorities

Questo metodo invoca:

- controlla i permessi dell'utente tramite l'invocazione del metodo `getAdministrator` del campo `account`;
- se è un amministratore, restituisce una lista dei permessi `ProdulyticsGrantedAuthority.ACCOUNT` e `ProdulyticsGrantedAuthority.ADMIN`;
- altrimenti, restituisce una lista con `ProdulyticsGrantedAuthority.ACCOUNT`.

5.2.44.4 isAccountNonExpired

Override del metodo dell'interfaccia `UserDetails`: restituisce sempre `true` perché la funzionalità non è implementata.

5.2.44.5 isAccountNonLocked

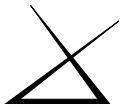
Override del metodo dell'interfaccia `UserDetails`: restituisce sempre `true` perché la funzionalità non è implementata.

5.2.44.6 isCredentialsNonExpired

Override del metodo dell'interfaccia `UserDetails`: restituisce sempre `true` perché la funzionalità non è implementata.

5.2.44.7 isEnabled

Override del metodo dell'interfaccia `UserDetails`: restituisce `true` se l'utente non è archiviato, `false` altrimenti.

**5.2.45 ProdulyticsGrantedAuthority****5.2.45.1 getAuthority**

Questo metodo restituisce i valori dell'enumerazione, ADMIN e ACCOUNT, in una stringa.

5.2.46 AccountsConfiguration**5.2.46.1 findAccountUseCase**

Questo metodo deve creare una nuova istanza di `FindAccountService` usando i parametri ricevuti.

5.2.46.2 updateAccountPasswordUseCase

Questo metodo deve creare una nuova istanza di `UpdateAccountPasswordService` usando i parametri ricevuti.

5.2.47 AdminsAccountsConfiguration**5.2.47.1 getTinyAccountsUseCase**

Questo metodo deve creare una nuova istanza di `GetTinyAccountsService` usando i parametri ricevuti.

5.2.47.2 insertAccountUseCase

Questo metodo deve creare una nuova istanza di `InsertAccountService` usando i parametri ricevuti.

5.2.47.3 updateAccountArchiveStatusUseCase

Questo metodo deve creare una nuova istanza di `UpdateAccountArchiveStatusService` usando i parametri ricevuti.

5.2.47.4 updateAccountByAdminUseCase

Questo metodo deve creare una nuova istanza di `UpdateAccountByAdminService` usando i parametri ricevuti.

5.2.48 AdminsDevicesConfiguration**5.2.48.1 getCharacteristicsUseCase**

Questo metodo deve creare una nuova istanza di `GetCharacteristicsService` usando i parametri ricevuti.

5.2.48.2 getDeviceDetailsUseCase

Questo metodo deve creare una nuova istanza di `GetDeviceDetailsService` usando i parametri ricevuti.

5.2.48.3 getDevicesUseCase

Questo metodo deve creare una nuova istanza di `GetDevicesService` usando i parametri ricevuti.

5.2.48.4 insertCharacteristicUseCase

Questo metodo deve creare una nuova istanza di `InsertCharacteristicService` usando i parametri ricevuti.

5.2.48.5 insertDeviceUseCase

Questo metodo deve creare una nuova istanza di `InsertDeviceService` usando i parametri ricevuti.

5.2.48.6 updateCharacteristicArchiveStatusUseCase

Questo metodo deve creare una nuova istanza di `UpdateCharacteristicArchiveStatusService` usando i parametri ricevuti.

5.2.48.7 updateCharacteristicUseCase

Questo metodo deve creare una nuova istanza di `UpdateCharacteristicService` usando i parametri ricevuti.

5.2.48.8 updateDeviceArchiveStatusUseCase

Questo metodo deve creare una nuova istanza di `UpdateDeviceArchiveStatusService` usando i parametri ricevuti.

5.2.48.9 updateDeviceDeactivateStatusUseCase

Questo metodo deve creare una nuova istanza di `UpdateDeviceDeactivateStatusService` usando i parametri ricevuti.

5.2.48.10 updateDeviceNameUseCase

Questo metodo deve creare una nuova istanza di `UpdateDeviceNameService` usando i parametri ricevuti.

5.2.49 DetectionsConfiguration

5.2.49.1 getDetectionsUseCase

Questo metodo deve creare una nuova istanza di `GetDetectionsService` usando i parametri ricevuti.

5.2.50 DevicesConfiguration

5.2.50.1 getLimitsUseCase

Questo metodo deve creare una nuova istanza di `GetLimitsService` usando i parametri ricevuti.

5.2.50.2 getUnarchivedCharacteristicsUseCase

Questo metodo deve creare una nuova istanza di `GetUnarchivedCharacteristicsService` usando i parametri ricevuti.

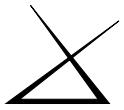
5.2.50.3 getUnarchivedDevices

Questo metodo deve creare una nuova istanza di `GetUnarchivedDevicesService` usando i parametri ricevuti.

5.2.51 EncoderConfig

5.2.51.1 getEncoder

Questo metodo restituisce un'istanza della classe `BCryptPasswordEncoder`.



5.2.52 SecurityConfiguration

5.2.52.1 authenticationProvider

Questo metodo:

- crea una nuova istanza di `DaoAuthenticationProvider` e la salva in `provider`;
- imposta i dettagli degli utenti contenuti nel campo `customAccountDetailsService` al `provider`, invocando il suo metodo `setUserDetailsService`;
- imposta il cifratore da usare invocando il metodo `getEncoder` del campo `encoder` e passando il risultato al metodo `setPasswordEncoder` del `provider`;
- infine, ritorna il `provider`.

5.2.52.2 configure

Questo metodo:

- permette a chiunque di effettuare richieste all'endpoint `/`;
- permette agli amministratori di effettuare richieste all'endpoint `/admin/**`;
- permette agli utenti di effettuare richieste all'endpoint `/**`;
- inoltre, se arriva una richiesta con opzione di memorizzazione della sessione, genera un cookie per memorizzare la sessione.

5.2.53 UserDetailsAdapterConfiguration

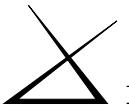
5.2.53.1 userDetailsService

Questo metodo deve creare una nuova istanza di `UserDetailsAdapter` usando i parametri ricevuti.

5.2.54 Diagrammi di sequenza

Nei seguenti diagrammi sono presenti degli elementi comuni, ossia:

- **Spring Boot Framework**, con cui si intende una rappresentazione simbolica del framework Spring Boot, che si occupa di mediare tra le richieste HTTP ricevute e le classi di tipo `Controller`;
- **Spring Impl NomeRepository**, con cui si intende una rappresentazione simbolica dell'implementazione di Spring del repository, dove “Nome” può essere “`Account`”, “`Characteristic`”, “`Detection`” o “`Device`”.



5.2.54.1 Modifica password

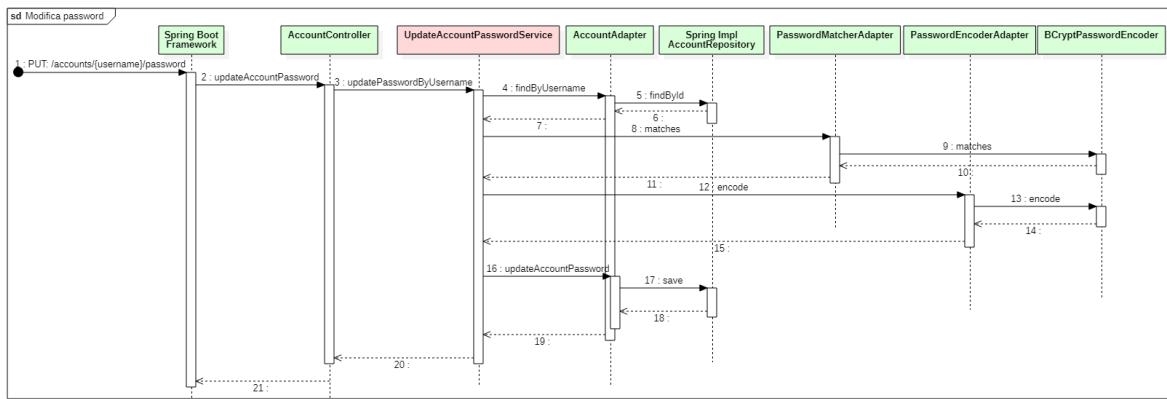


Figura 66: Diagramma di sequenza relativo alla modifica della password.

Questo diagramma illustra la gestione a runtime del caso di successo della modifica della propria password, svolta da un utente. In particolare:

- **AccountController** riceve la richiesta e la inoltra a **UpdateAccountPasswordService**;
- il service verifica che l'utente esista tramite una richiesta a **AccountAdapter**, che recupera l'informazione dal repository;
- il service verifica che la password corrente che è stata inserita sia corretta tramite una richiesta a **PasswordMatcherAdapter**, che usa un'istanza di **BCryptPasswordEncoder** per soddisfarla;
- il service cifra la nuova password tramite una richiesta a **PasswordEncoderAdapter**, che usa un'istanza di **BCryptPasswordEncoder** per soddisfarla;
- il service manda una richiesta ad **AccountAdapter** per memorizzare le modifiche effettuate.

5.2.54.2 Lista macchine non archiviate

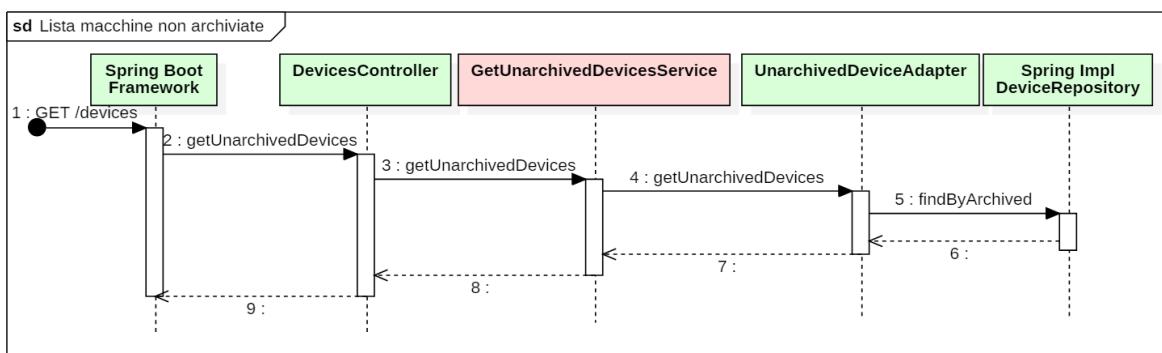


Figura 67: Diagramma di sequenza relativo all'ottenimento delle macchine non archiviate.

Questo diagramma illustra la gestione a runtime del caso di successo di ottenimento della lista delle macchine non archiviate. In particolare:

- **DevicesController** riceve la richiesta e la inoltra a **GetUnarchivedDevicesService**;
- il service inoltra la richiesta a **UnarchivedDeviceAdapter**, che recupera l'informazione dal repository;
- l'informazione viene ritornata al richiedente.

5.2.54.3 Lista caratteristiche non archivate di una macchina

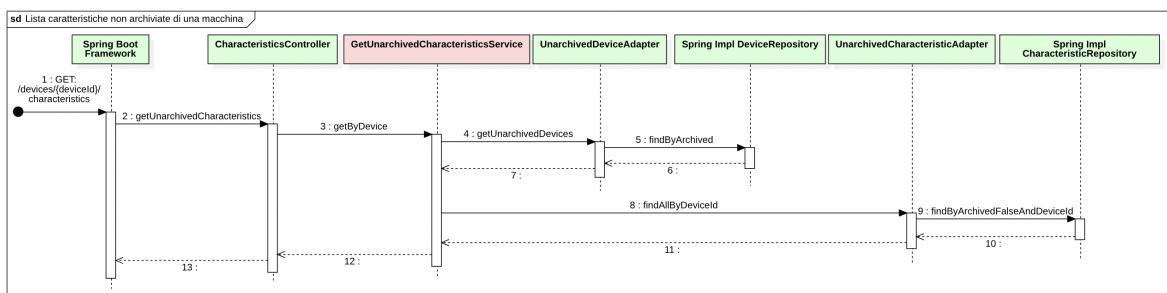


Figura 68: Diagramma di sequenza relativo all'ottenimento della lista delle caratteristiche non archivate di una macchina.

Questo diagramma illustra la gestione a runtime del caso di successo di ottenimento delle caratteristiche di una macchina. In particolare:

- **CharacteristicsController** riceve la richiesta e la inoltra a **GetUnarchivedCharacteristicsService**;
- il service verifica che la macchina esista tramite una richiesta a **UnarchivedDeviceAdapter**, che controlla nel repository;
- il service manda una richiesta a **UnarchivedCharacteristicAdapter** per ottenere le caratteristiche, che recupera le informazioni dal repository;
- le informazioni vengono ritornate al richiedente.

5.2.54.4 Rilevazioni di una caratteristica

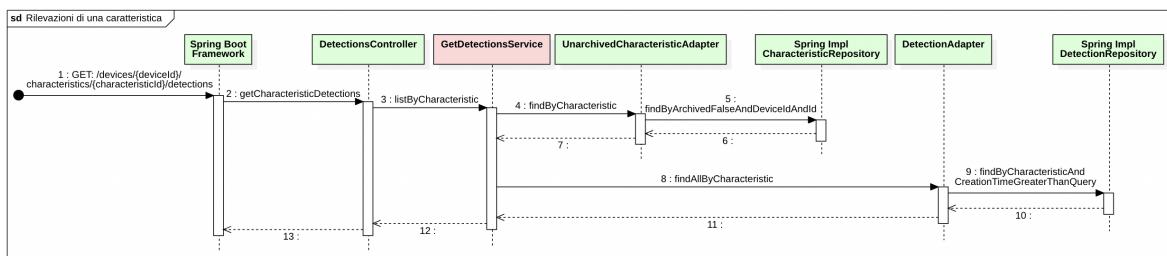


Figura 69: Diagramma di sequenza relativo all'ottenimento dei limiti di una caratteristica.

Questo diagramma illustra la gestione a runtime del caso di successo di ottenimento delle rilevazioni di una caratteristica. In particolare:

- **DetectionsController** riceve la richiesta e la inoltra a **GetDetectionsService**;
- il service verifica che la caratteristica esista tramite una richiesta a **UnarchivedCharacteristicAdapter**, che controlla nel repository;
- il service manda una richiesta a **DetectionAdapter** per ottenere le rilevazioni, che recupera le informazioni dal repository;
- le informazioni vengono ritornate al richiedente.

5.2.54.5 Limiti e media di una caratteristica

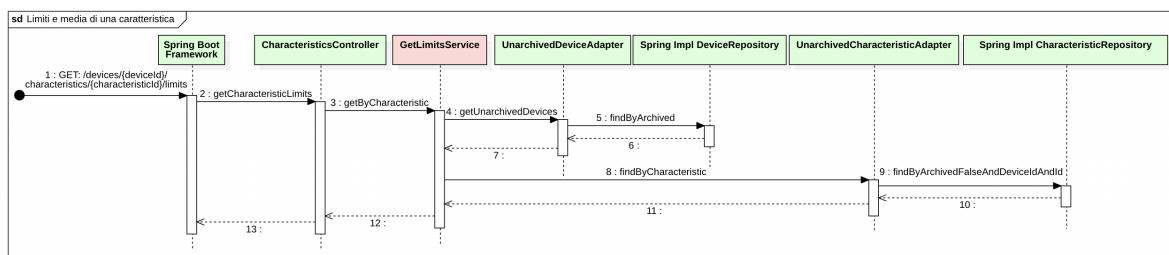
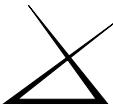


Figura 70: Diagramma di sequenza relativo all'ottenimento dei limiti di una caratteristica.

Questo diagramma illustra la gestione a runtime del caso di successo di ottenimento dei limiti tecnici e della media della caratteristica di una macchina. In particolare:

- **CharacteristicsController** riceve la richiesta e la inoltra a **GetLimitsService**;
- il service verifica che la macchina esista tramite una richiesta a **UnarchivedDeviceAdapter**, che controlla nel repository;
- il service inoltra la richiesta a **UnarchivedCharacteristicAdapter**, che recupera le informazioni dal repository;
- le informazioni vengono ritornate al richiedente.



5.2.54.6 Lista macchine

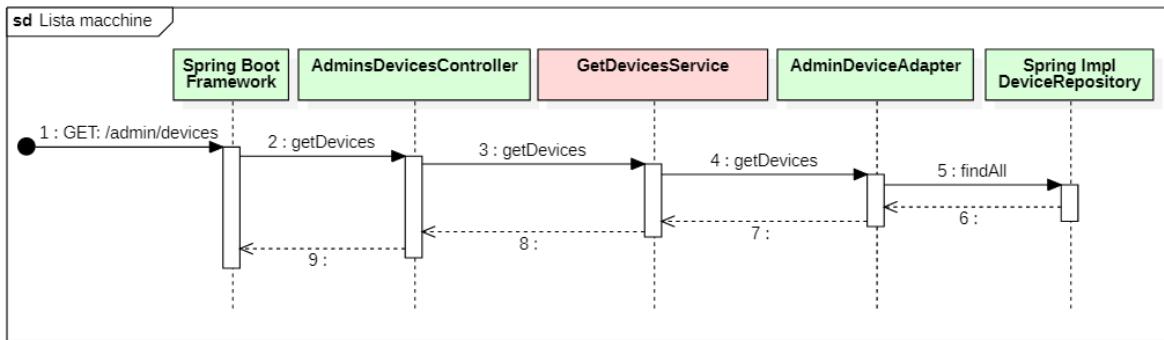


Figura 71: Diagramma di sequenza relativo all'ottenimento della lista delle macchine.

Questo diagramma illustra la gestione a runtime del caso di successo di ottenimento della lista delle macchine, richiesta da un amministratore. In particolare:

- **AdminsDevicesController** riceve la richiesta e la inoltra a **GetDevicesService**;
- il service inoltra la richiesta a **AdminDeviceAdapter**, che recupera le macchine dal repository;
- il risultato viene ritornato al richiedente.

5.2.54.7 Archiviazione macchina

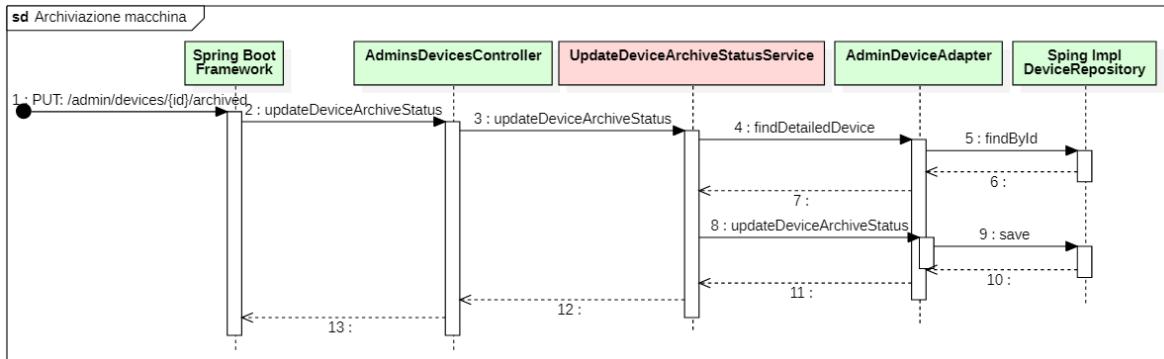
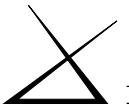


Figura 72: Diagramma di sequenza relativo all'archiviazione di una macchina.

Questo diagramma illustra la gestione a runtime del caso di successo del cambiamento dello stato di archiviazione di una macchina, richiesto da un amministratore. In particolare:

- **AdminsDevicesController** riceve la richiesta e la inoltra a **UpdateDeviceArchiveStatusService**;
- il service verifica che la macchina esista tramite una richiesta a **AdminDeviceAdapter**, che recupera la macchina dal repository;
- il service inoltra la richiesta di modifica dello stato di archiviazione a **AdminDeviceAdapter**, che memorizza il cambiamento nel repository.



5.2.54.8 Disattivazione macchina

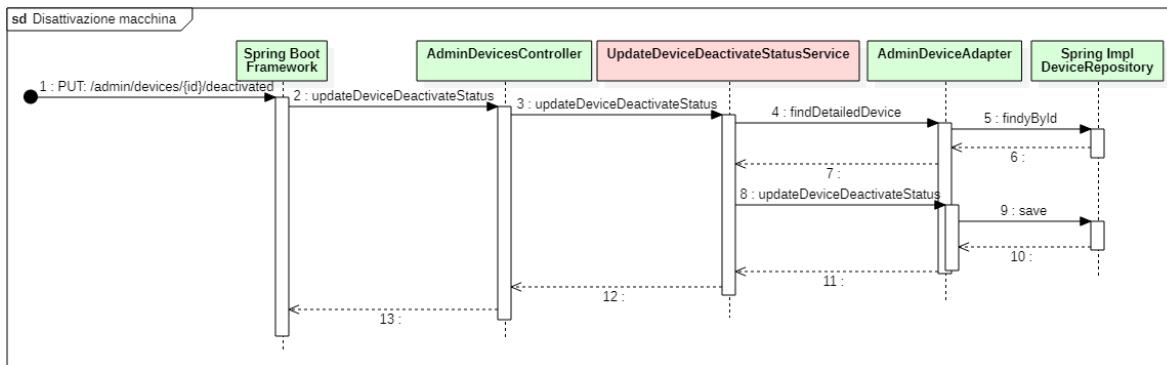


Figura 73: Diagramma di sequenza relativo alla disattivazione di una macchina.

Questo diagramma illustra la gestione a runtime del caso di successo del cambiamento dello stato di attivazione di una macchina, richiesto da un amministratore. In particolare:

- **AdminsDevicesController** riceve la richiesta e la inoltra a **UpdateDeviceDeactivateStatusService**;
- il service verifica che la macchina esista tramite una richiesta a **AdminDeviceAdapter**, che recupera la macchina dal repository;
- il service inoltra la richiesta di modifica dello stato di attivazione a **AdminDeviceAdapter**, che memorizza il cambiamento nel repository.

5.2.54.9 Dettagli macchina

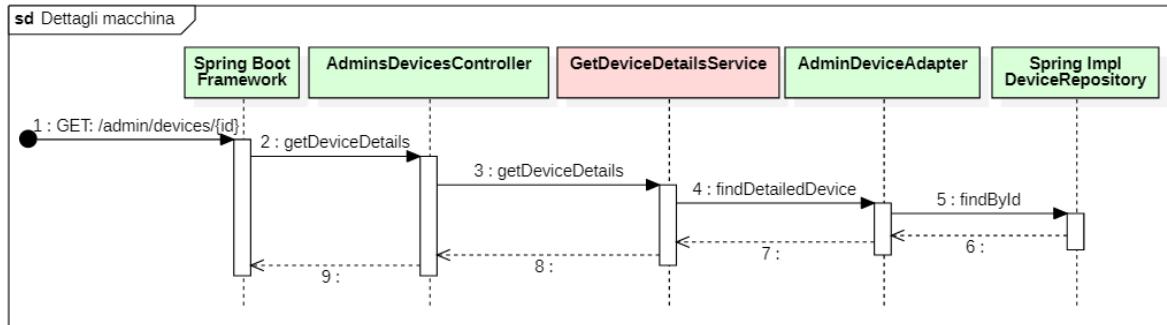
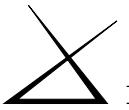


Figura 74: Diagramma di sequenza relativo all'ottenimento dei dettagli di una macchina.

Questo diagramma illustra la gestione a runtime del caso di successo dell'ottenimento dei dettagli di una macchina, richiesto da un amministratore. In particolare:

- **AdminsDevicesController** riceve la richiesta e la inoltra a **GetDeviceDetailsService**;
- il service inoltra la richiesta a **AdminDeviceAdapter**, che recupera la macchina dal repository;
- l'informazione viene ritornata al richiedente.



5.2.54.10 Modifica macchina

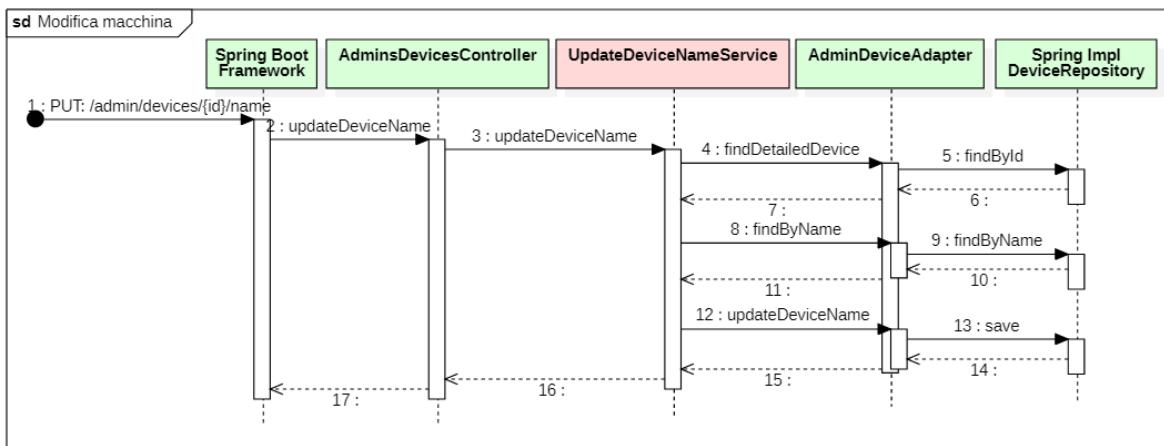


Figura 75: Diagramma di sequenza relativo alla modifica di una macchina.

Questo diagramma illustra la gestione a runtime del caso di successo della modifica del nome di una macchina, richiesta da un amministratore. In particolare:

- **AdminsDevicesController** riceve la richiesta e la inoltra a **UpdateDeviceNameService**;
- il service verifica che la macchina esista tramite una richiesta a **AdminDeviceAdapter**, che recupera la macchina dal repository;
- il service verifica che non esista già una macchina con lo stesso nome tramite una richiesta a **AdminDeviceAdapter**, che controlla nel repository;
- il service manda una richiesta ad **AccountAdapter** per memorizzare la modifica effettuata.

5.2.54.11 Inserimento macchina

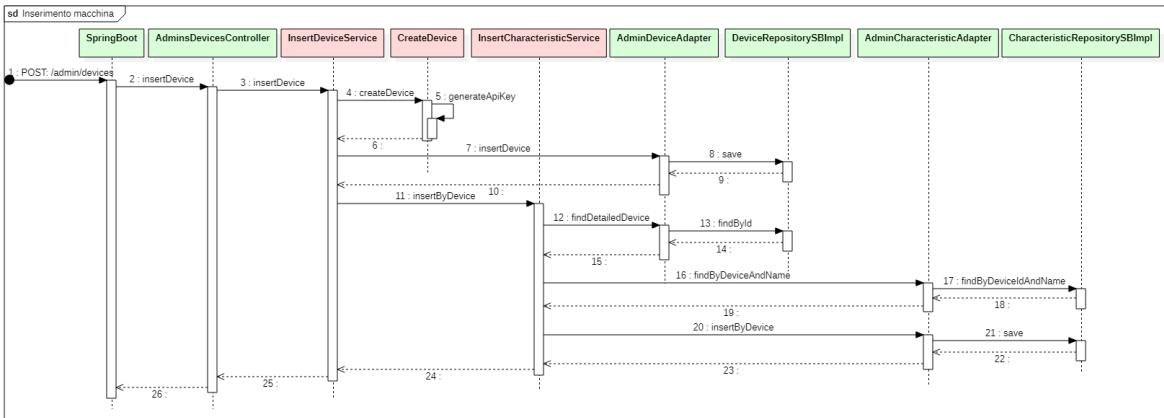
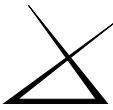


Figura 76: Diagramma di sequenza relativo all'inserimento di una macchina.



Questo diagramma illustra la gestione a runtime del caso di successo di inserimento di una macchina, richiesta da un amministratore. In particolare:

- **AdminsDevicesController** riceve la richiesta e la inoltra a **InsertDeviceService**;
- il service verifica che non esista già una macchina con lo stesso nome tramite una richiesta a **AdminDeviceAdapter**, che controlla nel repository;
- il service manda una richiesta a **CreateDevice** per creare una nuova macchina, che gli viene restituita;
- il service manda una richiesta a **AccountAdapter** per memorizzare la macchina appena creata;
- il service manda una richiesta a **InsertCharacteristicService** per inserire le caratteristiche nella macchina appena creata;
- **InsertCharacteristicService** verifica che la macchina esista tramite una richiesta a **AdminDeviceAdapter**, che controlla nel repository;
- **InsertCharacteristicService** verifica che non esista già una caratteristica con lo stesso nome tramite una richiesta a **AdminCharacteristicAdapter**, che controlla nel repository;
- **InsertCharacteristicService** verifica che i vincoli sui valori della caratteristica da inserire siano rispettati tramite l'interfaccia **CharacteristicConstraints**;
- **InsertCharacteristicService** invoca **AdminCharacteristicAdapter**, che memorizza la nuova caratteristica nel repository.

5.2.54.12 Lista delle caratteristiche di una macchina

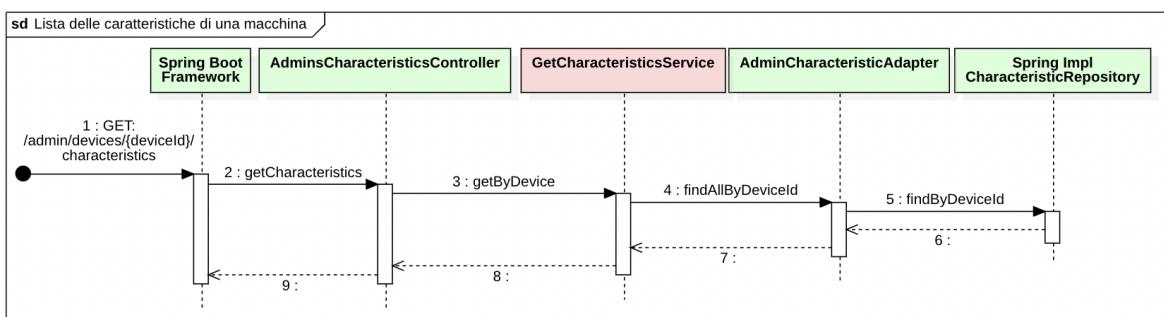


Figura 77: Diagramma di sequenza relativo all'ottenimento della lista delle caratteristiche di una macchina.

Questo diagramma illustra la gestione a runtime del caso di successo di inserimento di ottenimento delle caratteristiche di una macchina, richiesto da un amministratore. In particolare:

- **AdminsCharacteristicsController** riceve la richiesta e la inoltra a **GetCharacteristicsService**;
- il service inoltra la richiesta a **AdminCharacteristicAdapter**, che recupera le informazioni dal repository;
- le informazioni vengono ritornate al richiedente.

5.2.54.13 Archiviazione caratteristica

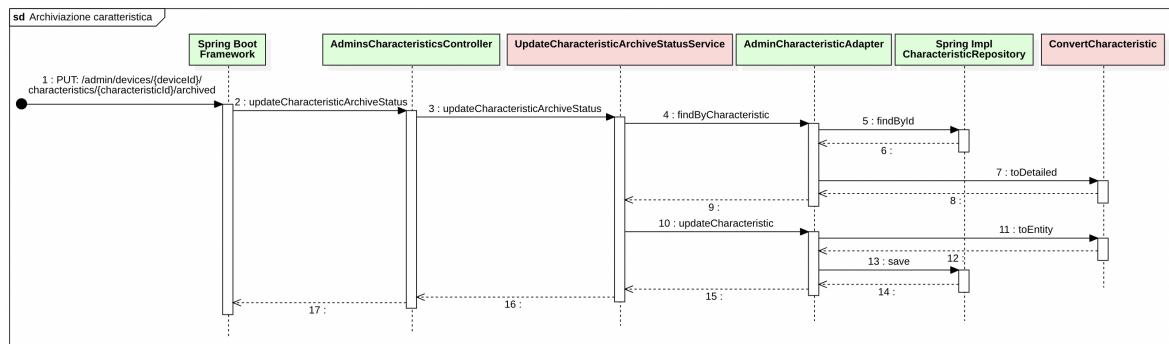


Figura 78: Diagramma di sequenza relativo all'archiviazione di una caratteristica.

Questo diagramma illustra la gestione a runtime del caso di successo della modifica dello stato di archiviazione di una caratteristica, richiesta da un amministratore. In particolare:

- `AdminsCharacteristicsController` riceve la richiesta e la inoltra a `UpdateCharacteristicArchiveStatusService`;
- il service verifica che la caratteristica esista tramite una richiesta a `AdminCharacteristicAdapter`, che controlla nel repository;
- il service manda una richiesta ad `AdminCharacteristicAdapter`, che memorizza la modifica nel repository.

5.2.54.14 Inserimento caratteristica

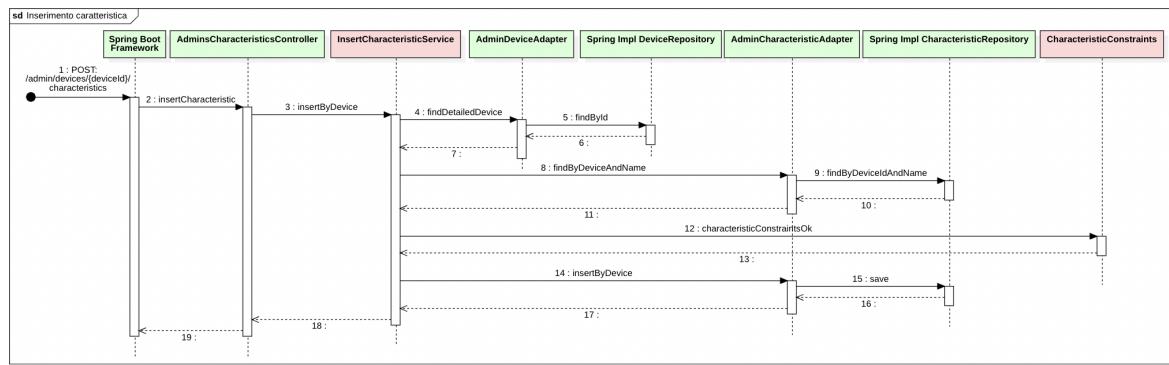


Figura 79: Diagramma di sequenza relativo all'inserimento di una caratteristica.

Questo diagramma illustra la gestione a runtime del caso di successo di inserimento di una nuova caratteristica in una macchina, richiesto da un amministratore. In particolare:

- `AdminsCharacteristicsController` riceve la richiesta e la inoltra a `InsertCharacteristicService`;

- il service verifica che la macchina esista tramite una richiesta a `AdminDeviceAdapter`, che controlla nel repository;
- il service verifica che non esista già una caratteristica con lo stesso nome tramite una richiesta a `AdminCharacteristicAdapter`, che controlla nel repository;
- il service verifica che i vincoli sui valori della caratteristica da inserire siano rispettati tramite l'interfaccia `CharacteristicConstraints`;
- il service manda una richiesta ad `AdminCharacteristicAdapter`, che memorizza la nuova caratteristica nel repository.

5.2.54.15 Modifica caratteristica

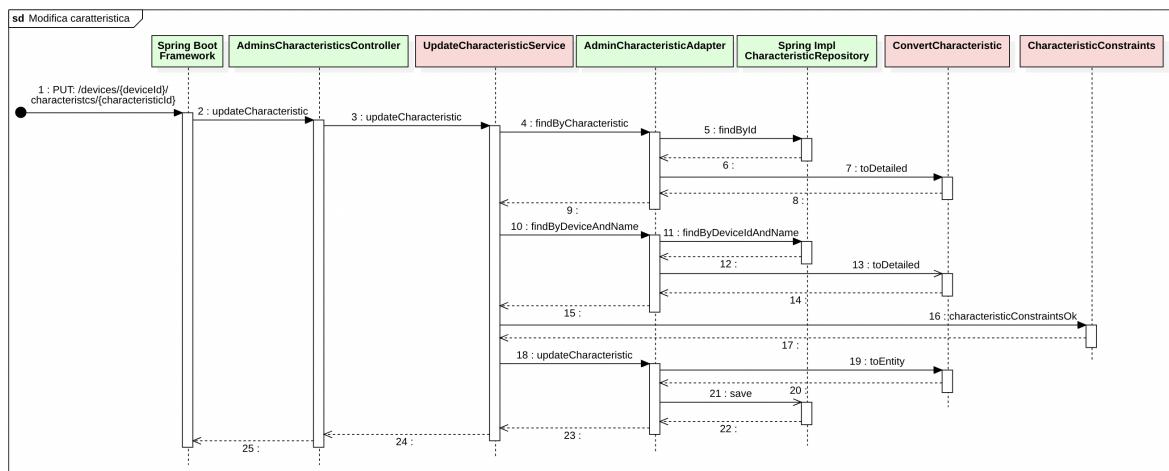
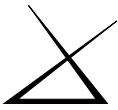


Figura 80: Diagramma di sequenza relativo alla modifica di una caratteristica.

Questo diagramma illustra la gestione a runtime del caso di successo di modifica di una caratteristica, richiesta da un amministratore. In particolare:

- `AdminsCharacteristicsController` riceve la richiesta e la inoltra a `UpdateCharacteristicService`;
- il service verifica che la caratteristica esista tramite una richiesta a `AdminCharacteristicAdapter`, che controlla nel repository;
- il service verifica che non esista già una caratteristica con il nuovo nome tramite una richiesta a `AdminCharacteristicAdapter`, che controlla nel repository;
- il service verifica che i vincoli sui valori della caratteristica modificata siano rispettati tramite l'interfaccia `CharacteristicConstraints`;
- il service manda una richiesta ad `AdminCharacteristicAdapter` per memorizzare la modifica.



5.2.54.16 Lista utenti

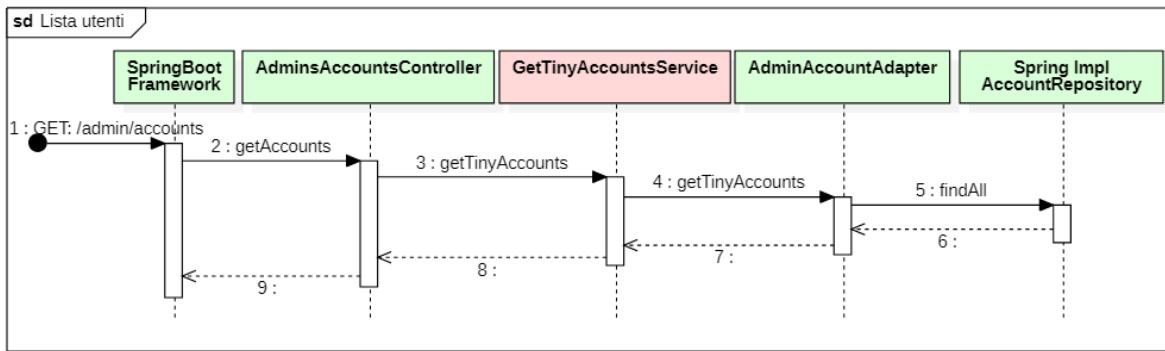


Figura 81: Diagramma di sequenza relativo all'ottenimento della lista degli utenti.

Questo diagramma illustra la gestione a runtime del caso di successo di ottenimento della lista degli utenti, richiesta da un amministratore. In particolare:

- AdminsAccountsController riceve la richiesta e la inoltra a GetTinyAccountsService;
- il service inoltra la richiesta a AdminDeviceAdapter, che recupera gli utenti dal repository;
- il risultato viene ritornato al richiedente.

5.2.54.17 Archiviazione utente

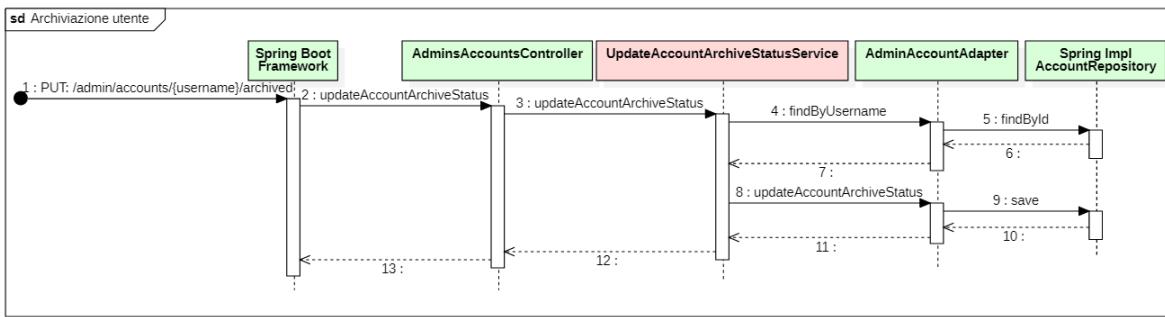
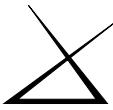


Figura 82: Diagramma di sequenza relativo all'archiviazione di un utente.

Questo diagramma illustra la gestione a runtime del caso di successo del cambiamento dello stato di archiviazione di un utente, richiesto da un amministratore. In particolare:

- AdminsAccountsController riceve la richiesta e la inoltra a UpdateAccountArchiveStatusService;
- il service verifica che l'utente esista tramite una richiesta a AdminAccountAdapter, che recupera l'utente dal repository;
- il service inoltra la richiesta di modifica dello stato di archiviazione a AdminAccountAdapter, che memorizza il cambiamento nel repository.



5.2.54.18 Inserimento utente

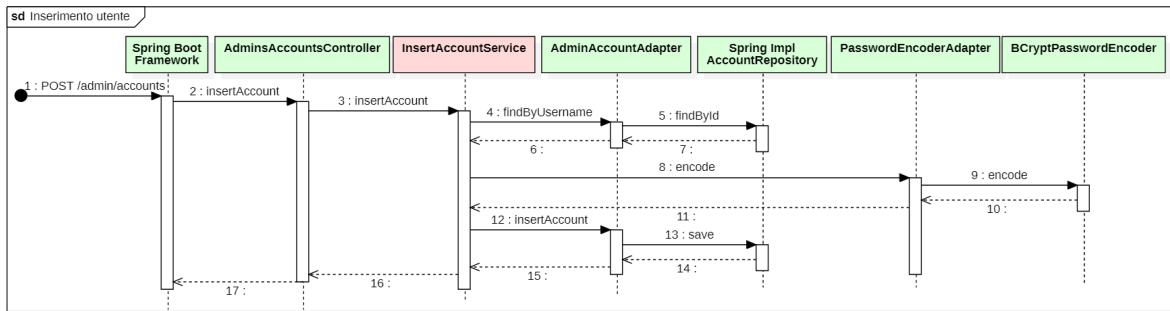


Figura 83: Diagramma di sequenza relativo all'inserimento di un utente.

Questo diagramma illustra la gestione a runtime del caso di successo di inserimento di un utente, richiesta da un amministratore. In particolare:

- `AdminsAccountsController` riceve la richiesta e la inoltra a `InsertAccountService`;
- il service verifica che non esista già un utente con lo stesso username tramite una richiesta a `AdminDeviceAdapter`, che controlla nel repository;
- il service cifra la password tramite una richiesta a `PasswordEncoderAdapter`, che usa un'istanza di `BCryptPasswordEncoder` per soddisfarla;
- il service manda una richiesta ad `AdminAccountAdapter` per memorizzare l'utente.

5.2.54.19 Modifica utente

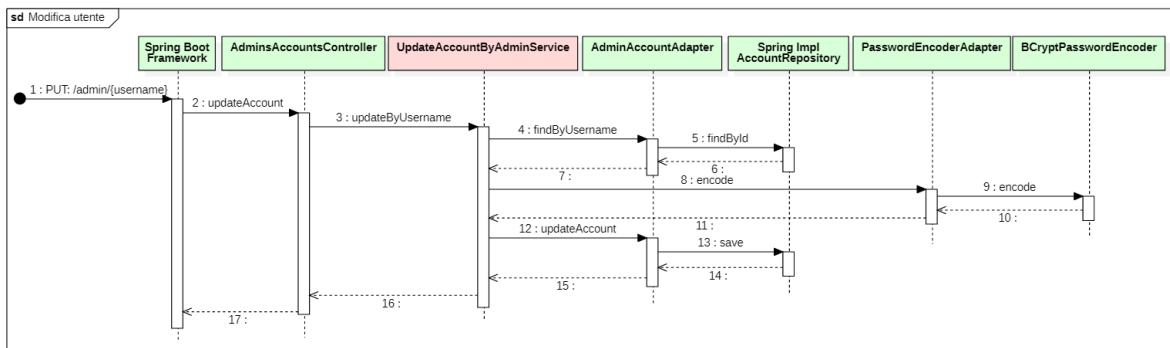


Figura 84: Diagramma di sequenza relativo alla modifica di un utente da parte di un amministratore.

Questo diagramma illustra la gestione a runtime del caso di successo della modifica dei dati di un utente, richiesta da un amministratore. In particolare:

- `AdminsAccountsController` riceve la richiesta e la inoltra a `UpdateAcciuntByAdminService`;

- il service verifica che l'utente esista tramite una richiesta a `AdminAccountAdapter`, che recupera l'utente dal repository;
- il service cifra la nuova password tramite una richiesta a `PasswordEncoderAdapter`, che usa un'istanza di `BCryptPasswordEncoder` per soddisfarla;
- il service manda una richiesta ad `AccountAdapter` per memorizzare la modifica effettuata.

5.2.54.20 Gestione di un errore

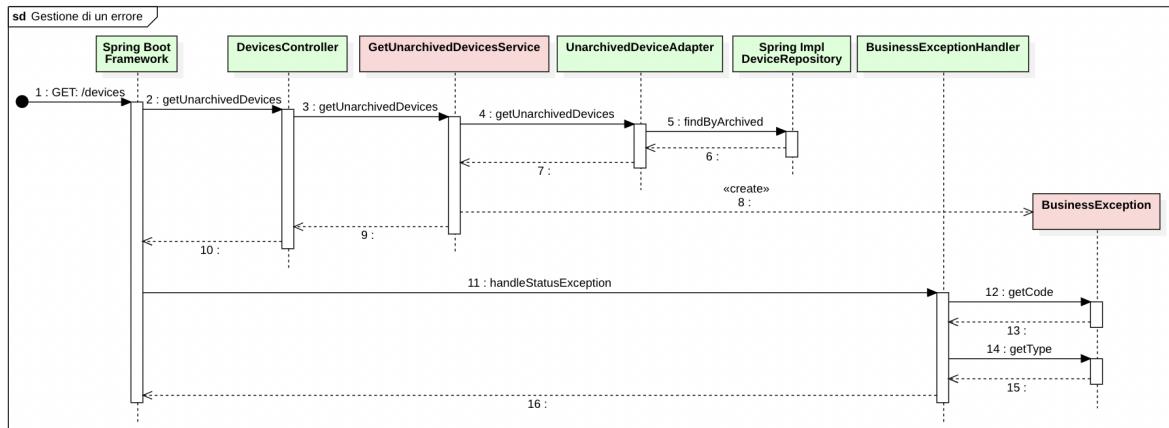


Figura 85: Diagramma di sequenza relativo alla gestione di un errore.

5.3 API rilevazioni

5.3.1 DetectionsController

5.3.1.1 addDetection

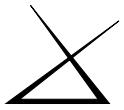
Questo metodo si occupa solo di chiamare il metodo `processIncomingDetection` del campo `processIncomingDetectionUseCase`, passandogli la rilevazione in arrivo che ha appena ricevuto.

5.3.2 BusinessExceptionHandler

5.3.2.1 handleStatusException

Questo metodo si occupa di produrre una risposta di errore in base all'eccezione lanciata. Deve quindi:

- individuare il codice di stato HTTP, ovvero un oggetto di tipo `HttpStatus` in base al tipo di errore, che può ottenere tramite `BusinessException::getType`, in particolare:
 - `ErrorType.AUTHENTICATION` corrisponde al codice 401, ovvero `HttpStatus.UNAUTHORIZED`;
 - `ErrorType.NOT_FOUND` corrisponde al codice 404, ovvero `HttpStatus.NOT_FOUND`;
 - `ErrorType.ARCHIVED` corrisponde al codice 410, ovvero `HttpStatus.GONE`.
- creare il corpo della risposta, ovvero creare una mappa che associa la stringa "errorCode" al codice di errore ottenuto tramite `BusinessException::getCode`;
- creare l'oggetto di risposta, ovvero un `ResponseEntity`, con i due valori appena individuati.



5.3.3 BusinessException

5.3.3.1 getCode

Questo metodo deve ritornare il valore del campo `code`.

5.3.3.2 getType

Questo metodo deve ritornare il valore del campo `type`.

5.3.4 DetectionsAdapter

5.3.4.1 findDeviceByApiKey

Questo metodo deve utilizzare il metodo `findByApiKey` del campo `deviceRepository` per ottenere l'entità `DeviceEntity` con la chiave API passata. Se essa è presente dovrà essere mappata in una istanza di `DeviceInfo` utilizzando `Optional::map`.

5.3.4.2 findCharacteristicByName

Questo metodo deve utilizzare il metodo `findByIdAndName` del campo `characteristicRepository` per ottenere un'entità `CharacteristicEntity` con tale nome. Se essa è presente dovrà essere mappata in una istanza di `CharacteristicInfo` utilizzando `Optional::map`.

5.3.4.3 findLastDetections

Questo metodo deve utilizzare il metodo `findLastDetectionsById` del campo `detectionRepository` per ottenere una lista di entità `DetectionEntity` relative alla caratteristica interessata. Esse devono poi essere mappate in istanze di `Detection`, convertendo prima la lista in uno stream con `List::stream`, poi mappandole con `Stream::map`, e infine riconvertendo lo stream in una lista con `Stream::toList`.

5.3.4.4 insertDetection

Questo metodo deve utilizzare il metodo `save` del campo `detectionRepository` per memorizzare una nuova `DetectionEntity` che deve creare a partire dalla rilevazione ricevuta. Il campo `id` dell'entità deve essere `null`.

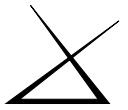
5.3.4.5 findLimits

Questo metodo deve utilizzare il metodo `findLimits` del campo `characteristicRepository` per ottenere i limiti tecnici, i limiti di processo e se l'auto-adjust è attivato. Deve poi combinare:

- i limiti tecnici in un'istanza di `TechnicalLimits`, se presenti, altrimenti `Optional.empty()`;
- i limiti di processo in un'istanza di `MeanStddev`, se l'auto-adjust è attivo, altrimenti `Optional.empty()`;
- i due valori appena individuati in un'istanza di `LimitsInfo`.

5.3.4.6 markOutlier

Questo metodo deve utilizzare il metodo `markOutlier` del campo `detectionRepository` per marcare la rilevazione ricevuta come anomala.



5.3.5 DetectionRepository

5.3.5.1 findLastDetectionsById

Questo metodo è implementato da Spring secondo la query passata all'annotazione `@Query`. Essa deve selezionare le ultime rilevazioni di una data caratteristica, dalla più nuova alla più vecchia, e prendere solo le ultime `count`. Successivamente deve invertire l'ordine, passando dalla più vecchia alla più recente.

5.3.5.2 markOutlier

Questo metodo è implementato da Spring secondo la query passata all'annotazione `@Query`. Essa deve aggiornare il campo `outlier` della `DetectionEntity` corrispondente alla `Detection` ricevuta come parametro, impostandolo a `true`.

5.3.6 CharacteristicRepository

5.3.6.1 findLimits

Questo metodo è implementato da Spring secondo la query passata all'annotazione `@Query`. Essa deve:

- ottenere il valore del campo `sample_size` della caratteristica;
- calcolare media e deviazione standard delle ultime `sample_size` (o 0, se non è impostato) rilevazioni della caratteristica;
- unire questi valori con i limiti tecnici della caratteristica;
- ritornare il risultato così ottenuto.

5.3.7 DetectionValidatorImpl

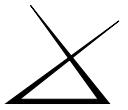
5.3.7.1 validateAndFindId

Questo metodo deve utilizzare il metodo `findDeviceByApiKey` del campo `findDeviceByApiKeyPort` per identificare la macchina data la sua chiave API e poi:

- se non è stata trovata deve lanciare una `BusinessException` con tipo `ErrorType.AUTHENTICATION` e codice "notAuthenticated";
- se è stata trovata ma è archiviata o disattivata deve lanciare una `BusinessException` con tipo `ErrorType.ARCHIVED` e codice "archived";
- altrimenti può continuare.

Deve poi utilizzare il metodo `findCharacteristicByName` del campo `findCharacteristicByNamePort` per identificare la caratteristica dato il suo nome e poi:

- se non è stata trovata deve lanciare una `BusinessException` con tipo `ErrorType.NOT_FOUND` e codice "characteristicNotFound";
- se è stata trovata ma è archiviata deve lanciare una `BusinessException` con tipo `ErrorType.ARCHIVED` e codice "archived";
- altrimenti può ritornare l'identificativo della caratteristica, formato a partire dalle informazioni della macchina e della caratteristica.



5.3.8 DetectionQueueImpl

5.3.8.1 DetectionQueueImpl

Questo costruttore deve assegnare la `DetectionSeriesFactory` presa come parametro al relativo campo e poi:

- inizializzare il campo `detectionsProcessor` con un `PublishProcessor` serializzato;
- inizializzare il campo `groupPhaser` con un nuovo `Phaser`;
- applicare un adattatore `groupBy` al campo `detectionsProcessor`, raggruppando secondo l'identificativo globale della caratteristica e inoltrando i gruppi con le loro chiavi al metodo privato `handleDetectionGroup`.

5.3.8.2 enqueueDetection

Questo metodo deve inviare la rilevazione al campo `detectionsProcessor` attraverso il suo metodo `onNext`.

5.3.8.3 close

Questo metodo deve chiudere il `detectionsProcessor` chiamando il suo metodo `onComplete` e poi aspettare che tutte le rilevazioni siano processate, registrandosi nel `groupPhaser` chiamando il suo metodo `register` e poi aspettando gli altri gruppi chiamando il suo metodo `arriveAndAwaitAdvance`.

5.3.8.4 handleDetectionGroup

```
private void handleDetectionGroup(  
    CharacteristicId key: l'identificativo globale della caratteristica;  
    Flowable<Detection> group: una serie di rilevazioni della caratteristica identificata da key da  
    analizzare.  
)
```

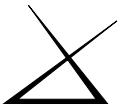
Questo metodo si occupa di gestire una serie di rilevazioni relative a una caratteristica. In particolare deve:

- segnalare la sua presenza chiamando il metodo `register` nel `groupPhaser`;
- creare una nuova `DetectionSeries` `serie` per la caratteristica a cui appartengono le rilevazioni, chiamando il metodo `createSeries` del campo `seriesFactory`;
- istruire `group` per:
 - analizzare gli elementi in un thread separato;
 - introdurre un timeout oltre al quale le risorse possono essere liberate;
 - segnalare al phaser chiamando `arriveAndDeregister` quando esso completa;
 - passare ogni rilevazione al metodo `insertDetection` di `series`.

5.3.9 DetectionSeriesImplFactory

5.3.9.1 createSeries

Questo metodo deve creare una nuova `DetectionSeriesImpl` usando i campi di questa classe e l'identificativo globale della caratteristica ricevuto come parametro.



5.3.10 DetectionSeriesImpl

5.3.10.1 insertDetection

Questo metodo deve:

- memorizzare nel repository la rilevazione ricevuta come parametro usando il metodo `insertDetection` del campo `ports`;
- ottenere i limiti di controllo attuali della caratteristica usando il metodo `calculateControlLimits` del campo `controlLimitsCalculator`;
- preparare una lista di rilevazioni esistenti da passare alle carte di controllo usando il metodo privato `detectionsForControlCharts`;
- analizzare le rilevazioni chiamando il metodo `analyzeDetections` del campo `controlChartsGroup`.

5.3.10.2 detectionsForControlChart

```
private List<? extends MarkableDetection> detectionsForControlCharts()
```

Questo metodo si occupa di preparare le ultime rilevazioni da passare alle carte di controllo. In particolare deve:

- calcolare il numero di rilevazioni da preparare, chiamando il metodo `requiredDetectionCount` del campo `controlChartsGroup`;
- ottenere le ultime rilevazioni chiamando il metodo `findLastDetections` del campo `ports`;
- mappare ogni rilevazione in una `MarkableDetectionAdapter` utilizzando uno `Stream`;
- ritornare la lista appena prodotta.

5.3.11 MarkableDetectionAdapter

5.3.11.1 value

Questo metodo deve ritornare il valore ritornato chiamando il metodo `value` del campo `detection`.

5.3.11.2 markOutlier

Questo metodo deve marcare la rilevazione rappresentata da questa classe chiamando il metodo `markOutlier` del campo `seriesPortFacade`.

5.3.12 ControlLimits

5.3.12.1 mean

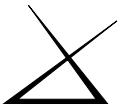
Questo metodo deve ritornare la metà della somma tra i campi `lowerLimit` e `upperLimit`.

5.3.12.2 lowerBCLimit

Questo metodo deve ritornare la differenza tra i valori ritornati dai metodi `mean` e `stddev`.

5.3.12.3 upperBCLimit

Questo metodo deve ritornare la somma tra i valori ritornati dai metodi `mean` e `stddev`.



5.3.12.4 lowerABLimit

Questo metodo deve ritornare la differenza tra il valore ritornato dal metodo `mean` e due volte il valore ritornato dal metodo `stddev`.

5.3.12.5 upperABLimit

Questo metodo deve ritornare la somma tra il valore ritornato dal metodo `mean` e due volte il valore ritornato dal metodo `stddev`.

5.3.12.6 stddev

`double stddev()`

Questo metodo deve ritornare un sesto della differenza tra i campi `upperLimit` e `lowerLimit`.

5.3.13 SeriesPortFacadeImpl

5.3.13.1 insertDetection

Questo metodo deve essere inoltrato al metodo `insertDetection` del campo `InsertDetectionPort`.

5.3.13.2 findLastDetections

Questo metodo deve essere inoltrato al metodo `findLastDetections` del campo `FindLastDetectionsPort`.

5.3.13.3 markOutlier

Questo metodo deve essere inoltrato al metodo `markOutlier` del campo `MarkOutlierPort`.

5.3.14 ControlLimitsCalculatorImpl

5.3.14.1 calculateControlLimits

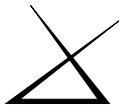
Questo metodo deve:

- ottenere le informazioni sui limiti chiamando il metodo `findLimits` del campo `findLimitsPort`;
- se le informazioni dei limiti includono informazioni sui limiti di processo, calcolare i limiti di controllo a partire da essi, chiamando il metodo `fromMeanStddev`;
- se invece le informazioni dei limiti includono solo informazioni sui limiti tecnici, calcolare i limiti di controllo a partire da essi, chiamando il metodo `fromTechnicalLimits`;
- se nessun tipo di limite è impostato allora si tratta di una situazione impossibile, lanciare una `RuntimeException` per segnalarlo.

5.3.14.2 fromMeanStddev

Questo metodo deve creare un'istanza di `ControlLimits` utilizzando:

- come limite inferiore la differenza tra la media e tre volte la deviazione standard dei limiti di processo;
- come limite superiore la somma tra la media e tre volte la deviazione standard dei limiti di processo.



5.3.14.3 fromTechnicalLimits

Questo metodo deve creare un'istanza di `ControlLimits` utilizzando:

- come limite inferiore il limite tecnico inferiore passato come parametro;
- come limite superiore il limite tecnico superiore passato come parametro.

5.3.15 ControlChartsGroupImpl

5.3.15.1 requiredDetectionCount

Questo metodo deve convertire il campo `controlCharts` in una lista, mappare ogni elemento al valore ritornato dal suo metodo `requiredDetectionCount` e infine ottenere il massimo. Se il massimo è `Optional.empty()`, ovvero non ci sono carte di controllo, allora ritorna 0.

5.3.15.2 analyzeDetections

Questo metodo deve, per ogni elemento del campo `controlCharts`:

- ottenere il valore ritornato dal suo metodo `requiredDetectionCount`;
- controllare se le rilevazioni ricevute sono abbastanza per tale metodo, altrimenti continuare con l'iterazione successiva;
- ottenere le ultime rilevazioni richieste dalla carta di controllo dell'iterazione corrente a partire da quelle ricevute come parametro, utilizzando il metodo `cutLastDetections`;
- passare le rilevazioni alla carta di controllo dell'iterazione corrente.

5.3.15.3 cutLastDetections

```
private List<? extends MarkableDetection> cutLastDetections(  
    List<? extends MarkableDetection> lastDetections: una lista di rilevazioni;  
    int count: il numero di ultime rilevazioni che si vuole ottenere.  
)
```

Questo metodo si occupa di ritornare le ultime `count` rilevazioni da una lista di rilevazioni. È richiesto `lastDetections.size() >= count`. In particolare deve utilizzare il metodo `subList`, passando come parametri `lastDetections.size() - count` e `listDetections.size()`.

5.3.16 ControlChartBeyondLimits

5.3.16.1 requiredDetectionCount

Questo metodo deve ritornare sempre 1.

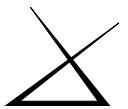
5.3.16.2 analyzeDetections

Questo metodo deve controllare se l'unica rilevazione ricevuta sia minore del limite di controllo inferiore o maggiore del limite di controllo superiore, e in tal caso chiamare il metodo `mark` su tale rilevazione.

5.3.17 ControlChartMixture

5.3.17.1 requiredDetectionCount

Questo metodo deve ritornare sempre 8.



5.3.17.2 analyzeDetections

Questo metodo deve controllare, utilizzando uno stream, se ogni rilevazione sia minore di `limits.lowerBCLimit()` o maggiore di `limits.upperBCLimit()`, e in tal caso marcarle tutte come anomale utilizzando il metodo `markAll` della classe `ControlChartUtils`.

5.3.18 ControlChartOverControl

5.3.18.1 requiredDetectionCount

Questo metodo deve ritornare sempre 14.

5.3.18.2 analyzeDetections

Questo metodo deve controllare se tutte le triplette di rilevazioni, ottenute con il metodo `windows` della classe `ControlChartUtils` passando `detections` e 3, sono in ordine crescente-decrescente o decrescente-crescente, e in tal caso marcare tutte le rilevazioni come anomale utilizzando il metodo `markAll` della classe `ControlChartUtils`.

5.3.19 ControlChartStratification

5.3.19.1 requiredDetectionCount

Questo metodo deve ritornare sempre 15.

5.3.19.2 analyzeDetections

Questo metodo deve controllare se tutte le rilevazioni hanno valore compreso tra `limits.lowerBCLimit()` e `limits.upperBCLimit()`, e in tal caso marcarle tutte come anomale utilizzando il metodo `markAll` della classe `ControlChartUtils`.

5.3.20 ControlChartTrend

5.3.20.1 requiredDetectionCount

Questo metodo deve ritornare sempre 7.

5.3.20.2 analyzeDetections

Questo metodo deve controllare se tutte le triplette di rilevazioni, ottenute con il metodo `windows` della classe `ControlChartUtils` passando `detections` e 3, sono in ordine crescente o decrescente, e in tal caso marcare tutte le rilevazioni come anomale utilizzando il metodo `markAll` della classe `ControlChartUtils`.

5.3.21 ControlChartZoneA

5.3.21.1 requiredDetectionCount

Questo metodo deve ritornare sempre 3.

5.3.21.2 analyzeDetections

Questo metodo deve:

- contare il numero di rilevazioni minore di `limits.lowerABLimit()`, utilizzando uno stream;
- contare il numero di rilevazioni maggiore di `limits.upperABLimit()`, utilizzando uno stream;

- se uno dei due numeri è maggiore o uguale a 2, marcare tutte le rilevazioni come anomale utilizzando il metodo `markAll` della classe `ControlChartUtils`.

5.3.22 ControlChartZoneB

5.3.22.1 requiredDetectionCount

Questo metodo deve ritornare sempre 5.

5.3.22.2 analyzeDetections

Questo metodo deve:

- contare il numero di rilevazioni minore di `limits.lowerBCLimit()`, utilizzando uno stream;
- contare il numero di rilevazioni maggiore di `limits.upperBCLimit()`, utilizzando uno stream;
- se uno dei due numeri è maggiore o uguale a 4, marcare tutte le rilevazioni come anomale utilizzando il metodo `markAll` della classe `ControlChartUtils`.

5.3.23 ControlChartZoneC

5.3.23.1 requiredDetectionCount

Questo metodo deve ritornare sempre 7.

5.3.23.2 analyzeDetections

Questo metodo deve:

- contare il numero di rilevazioni minore di `limits.mean()`, utilizzando uno stream;
- contare il numero di rilevazioni maggiore di `limits.mean()`, utilizzando uno stream;
- se uno dei due numeri è uguale a 7, marcare tutte le rilevazioni come anomale utilizzando il metodo `markAll` della classe `ControlChartUtils`.

5.3.24 ControlChartUtils

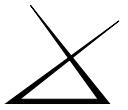
5.3.24.1 windows

Questo metodo deve:

- produrre uno `Stream` di indici corrispondenti al primo elemento della “finestra” usando il metodo `range` della classe `IntStream` e passando 0 come valore iniziale e `list.size() - size + 1` come valore finale;
- mappare ogni indice alla rispettiva finestra, utilizzando il metodo `subList` dell’interfaccia `list` e passando l’indice e l’indice incrementato di `size`;
- ritornare lo `Stream` così creato.

5.3.24.2 markAll

Questo metodo deve iterare tutte le rilevazioni e chiamare `markOutlier` su ognuna di esse.



5.3.25 DetectionsConfiguration

5.3.25.1 createDetectionValidator

Questo metodo deve creare una nuova istanza di `DetectionValidatorImpl` usando i parametri ricevuti.

5.3.25.2 createControlCharts

Questo metodo deve creare una nuova istanza di `ControlChartsImpl`, passando una lista contenente una nuova istanza di ogni classe che estende `ControlChart`.

5.3.25.3 createSeriesFacadePort

Questo metodo deve creare una nuova istanza di `SeriesFacadePortImpl` usando i parametri ricevuti.

5.3.25.4 controlLimitsCalculator

Questo metodo deve creare una nuova istanza di `ControlLimitsCalculatorImpl` usando i parametri ricevuti.

5.3.25.5 createDetectionSeriesFactory

Questo metodo deve creare una nuova istanza di `DetectionSeriesFactoryImpl` usando i parametri ricevuti.

5.3.25.6 createDetectionQueue

Questo metodo deve creare una nuova istanza di `DetectionQueueImpl` usando i parametri ricevuti.

5.3.25.7 createProcessDetectionUseCase

Questo metodo deve creare una nuova istanza di `DetectionsService` usando i parametri ricevuti.

5.3.26 Diagrammi di sequenza

5.3.26.1 Ricezione di una rilevazione

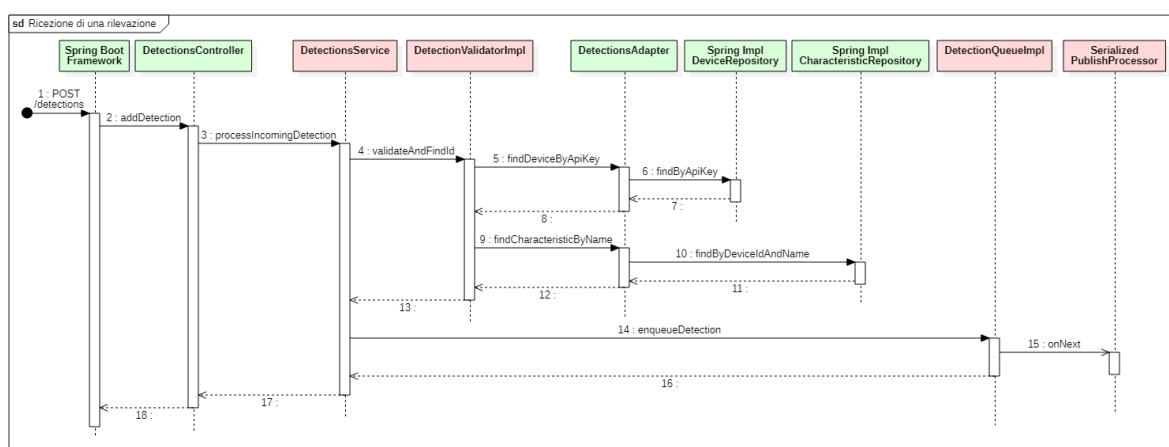


Figura 86: Diagramma di sequenza relativo alla ricezione di una rilevazione.

Questo diagramma illustra la gestione a runtime della ricezione di una rilevazione. In particolare:

- il framework Spring Boot, qui rappresentato simbolicamente come **Spring Boot Framework** si occupa di mediare tra la classe **DetectionsController** e le richieste HTTP;
- il **DetectionsController** delega l'implementazione al **DetectionsService**, il quale delega a **DetectionValidatorImpl** e **DetectionQueueImpl**;
- **DetectionValidatorImpl** utilizza il **DetectionAdapter** per comunicare con le implementazioni di Spring dei repository e validare la richiesta, che in questo caso è valida;
- **DetectionQueueImpl** invia la rilevazione a un **PublishProcessor** serializzato, la quale è un'operazione non bloccante.

5.3.26.2 Gestione di un errore

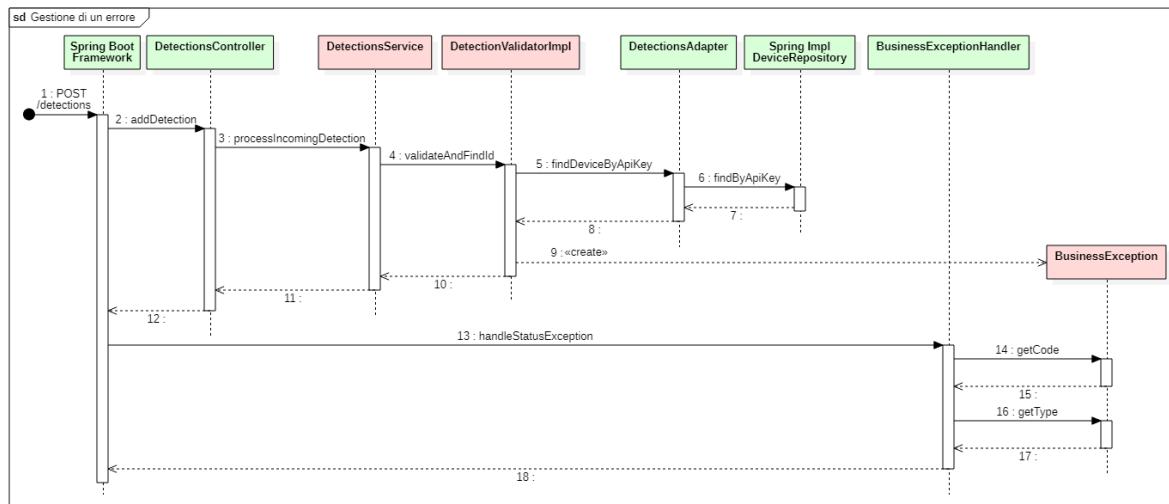


Figura 87: Diagramma di sequenza relativo alla gestione di un errore.

Questo diagramma illustra la gestione a runtime dell'analisi di una rilevazione. In particolare:

- in questo caso **DetectionValidatorImpl** determina che la rilevazione non è valida, lanciando quindi un'eccezione;
- l'eccezione si propaga fino al framework Spring Boot;
- esso invia la rilevazione a **BusinessExceptionHandler**, il quale produce una risposta di errore per l'utente.

5.3.26.3 Analisi di una rilevazione

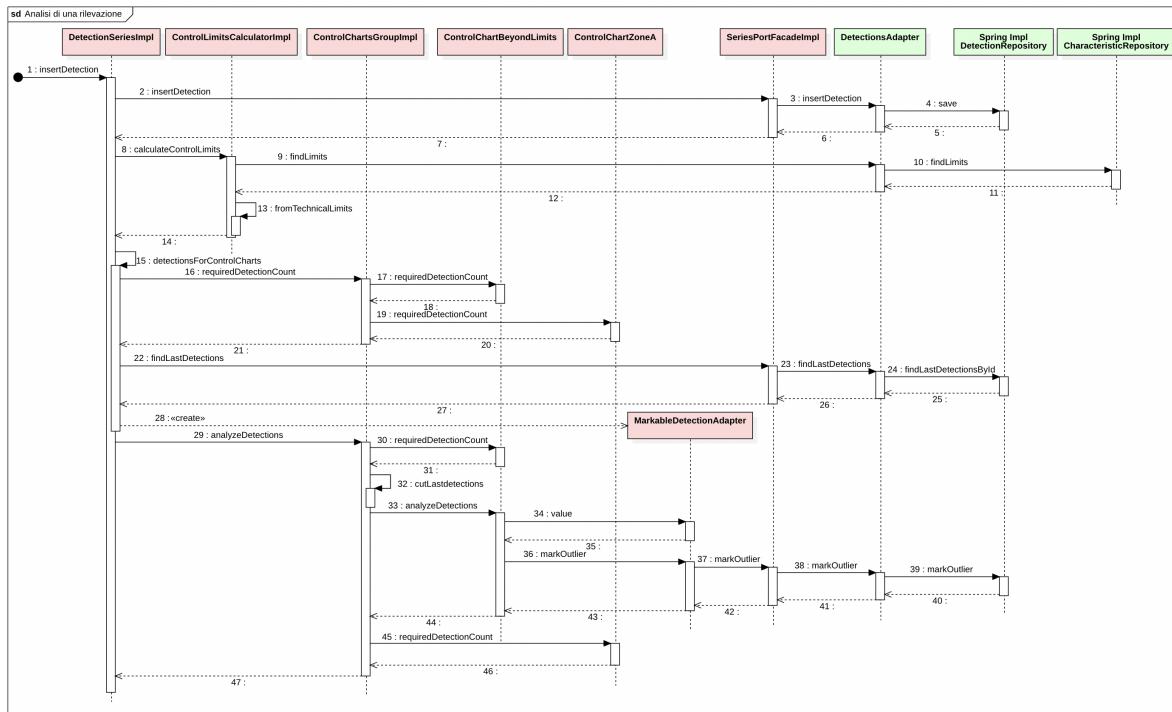
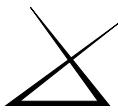


Figura 88: Diagramma di sequenza relativo all'analisi di una rilevazione.

Questo diagramma illustra la gestione a runtime dell'analisi di una rilevazione. In particolare:

- `DetectionSeriesImpl` memorizza la rilevazione, ottiene i limiti di controllo e ottiene le ultime rivelazioni necessarie alle carte di controllo;
- `DetectionSeriesImpl` converte le rilevazioni ottenute in rilevazioni marcabili per le carte di controllo, creando istanze di `MarkableDetectionAdapter`;
- `DetectionSeriesImpl` invia quindi le rilevazioni a `ControlChartsGroupImpl`, il quale le inoltra alle carte di controllo che contiene, in questo caso solo `ControlChartBeyondLimits` e `ControlChartZoneA`;
- in questo caso le rilevazioni ottenute sono solo una, abbastanza per `ControlChartBeyondLimits` ma non per `ControlChartZoneA`. Per questo motivo `analyzeDetections` viene chiamato solo su `ControlChartBeyondLimits`;
- in questo caso la rilevazione è oltre i limiti di controllo, e quindi `ControlChartBeyondLimits` la marca come anomala.



A Glossario

A

ACID

Acronimo di Atomicity, Consistency, Isolation, Durability. Indica le proprietà che, in ambito di $data\text{-}base_G$, una $transazione_G$ deve avere. Ha come priorità l'assicurare una struttura solida e affidabile.

Angular

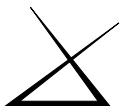
$Framework_G$ $JavaScript_G$ $open\ source_G$, sviluppato principalmente da Google, per lo sviluppo di $web\ application_G$ dinamiche.

API

Acronimo di Application Programming Interface, è un'insieme di procedure che offrono servizi per lo sviluppo e integrazione di sistemi software diversi.

Auto-adjust

Funzionalità relativa a una caratteristica di una macchina che permette di calcolare dinamicamente $media_G$ e $deviazione\ standard_G$, su un campione di dati forniti.

**B****Boilerplate**

Sezioni di codice ripetute, o riportate con poche variazioni, in svariati punti. Il codice boilerplate è difficile da manutenere, perché non ha una logica centralizzata.

Browser

Applicazione per l'acquisizione, la presentazione e la navigazione di risorse sul web.

C

Caratteristica

Attributo misurabile di una macchina, sul quale è possibile effettuare previsioni e calcoli statistici.

Carta di controllo

Strumento statistico per mantenere sotto controllo i parametri di un processo. In una carta di controllo sono presenti una $media_G$, due $limiti\ di\ controllo_G$ (inferiore e superiore), due $limiti\ di\ sorveglianza_G$ (inferiore e superiore), tre zone che dividono lo spazio in $zona\ A_G$, $zona\ B_G$ e $zona\ C_G$. In base al comportamento del processo rispetto ai limiti di controllo e alle zone si stabilisce se esso stia andando fuori controllo o meno. Per i dettagli sulle regole che stabiliscono se un processo produttivo stia andando fuori controllo consultare l'*Analisi dei Requisiti*.

ClickHouse

$DBMS_G$ *column-oriented* $_G$ e *open source* $_G$ specifico per $OLAP_G$.

Client

Componente software che fruisce di un servizio o di una risorsa offerti da un $server_G$.

Client-server

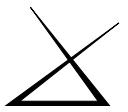
Architettura che identifica due categorie di componenti del sistema: i $server_G$, che mettono a disposizione determinati servizi, e i $client_G$, che ne usufruiscono. Questo tipo di architettura è comunemente associata ai sistemi distribuiti.

Column-oriented

Relativo a un $DBMS_G$, indica un $database_G$ che memorizza dati delle tabelle come sezioni di colonne piuttosto che di righe. Ciò porta vantaggi soprattutto alle operazioni di lettura.

Cypress

Software per l'automazione dei test di interfacce utente che agisce sul $browser_G$.



D

D3.js

Libreria $JavaScript_G$ per lo sviluppo di elementi grafici e interattivi in un $browser_G$. Utilizza funzioni JavaScript per selezionare elementi del DOM_G , creare elementi SVG, aggiungergli uno stile grafico, transizioni o altri effetti di movimento.

Database

Collezione organizzata di dati che risiede in un uno o più computer.

DBMS

Acronimo di DataBase Management System. È un software dedicato alla creazione, manipolazione e interrogazione efficiente di un $database_G$.

Deviazione standard

In statistica, è la misura di dispersione di un insieme di valori, più bassa è la deviazione standard più i valori si concentreranno intorno alla media. Si calcola con:

$$\sigma = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{N}}$$

dove \bar{x} è la $media_G$.

Docker

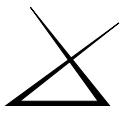
Software che automatizza il rilascio di applicazioni all'interno di contenitori software, fornendo quindi un livello di astrazione aggiuntivo.

Docker Compose

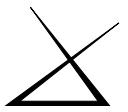
Docker Compose è uno strumento sviluppato per definire e condividere applicazioni multi-contenitore. Con Compose è possibile creare un file $YAML_G$ per definire i servizi dell'applicazione; inoltre è possibile creare, avviare e rimuovere tutti i servizi con un singolo comando.

DOM

Acronimo di Document Object Model, rappresenta una pagina web come un oggetto a struttura ad albero e consente ai linguaggi di programmazione web di modificarne dinamicamente la struttura e contenuto.

**E****Express**

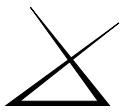
Express, o ExpressJS, è un *framework* per costruire servizi web con *Node.js*.

**F****Framework**

Architettura logica di supporto sulla quale un software può essere progettato e realizzato, facilitandone lo sviluppo. È definito da un insieme di classi astratte e dalle relazioni tra esse.

Front-end

Nello sviluppo web rappresenta tutto ciò che concerne la parte di presentazione dei dati all'utente.



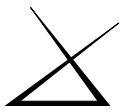
G

General purpose

Una tecnologia, come un linguaggio di programmazione o un $framework_G$, si dice “general purpose” se è utilizzabile per la creazione di applicazioni senza limitazioni di dominio.

GraphQL

Linguaggio di interrogazione e manipolazione dei dati $open\ source_G$ per API_G e un runtime per soddisfare $query_G$ con dati esistenti.



H

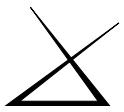
HTTP

Protocollo di comunicazione per lo scambio di ipertesti.

HTTP-status

Codici inviati da un $server_G$ in risposta a una richiesta $HTTP_G$ di un $client_G$, sono composti di tre cifre e indicano lo stato della richiesta:

- **1xx**: la richiesta è stata ricevuta;
- **2xx**: la richiesta è stata ricevuta e accettata;
- **3xx**: la richiesta necessita di un reindirizzamento per essere completata;
- **4xx**: la richiesta è malformata o non può essere soddisfatta;
- **5xx**: la richiesta è valida ma il server non è in grado di soddisfarla.

**J****Java**

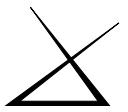
Linguaggio di programmazione orientato agli oggetti e interpretato dall'omonima macchina virtuale, ciò svincola il linguaggio da qualsiasi piattaforma hardware.

JavaScript

Linguaggio di programmazione web orientato agli oggetti ed eventi per la creazione di siti e applicazioni web dinamici e interattivi.

JSON

Acronimo di *JavaScript_G* Object Notation, è un formato di serializzazione basato su testo per lo scambio di dati, principalmente tra *server_G* e applicazione web.



L

Limiti di controllo

Relativamente alle *carte di controllo*_G, vengono individuati due valori: un limite superiore e un limite inferiore. Questi limiti definiscono la regione entro la quale un processo di produzione si trova sotto controllo. I limiti di controllo sono definiti dai *limiti tecnici*_G se la funzione di *auto-adjust*_G è disattivata, altrimenti corrispondono ai *limiti di processo*_G.

Limiti di processo

Sono i limiti calcolati dalla funzione di *auto-adjust*_G, di norma questi valori dovrebbero essere compresi nei *limiti tecnici*_G.

Limiti di riferimento

Sono i *limiti tecnici*_G se questi sono definiti, altrimenti corrispondono ai *limiti di processo*_G.

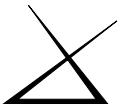
Limiti tecnici

Sono i limiti inferiore e superiore effettivi di una *macchina*_G.

Limiti di sorveglianza

Relativi alle *carte di controllo*_G, sono due: uno superiore e uno inferiore, posti rispettivamente a $\bar{x} - 2\sigma$ e $\bar{x} + 2\sigma$, dove \bar{x} è la *media*_G e σ la *deviazione standard*_G.

Sono i valori che limitano la *zona* B_G .



M

Macchina

Mezzo di produzione che si vuole monitorare.

Markup

Un linguaggio di markup è un insieme di regole che descrivono i meccanismi di rappresentazione o layout di un testo.

MariaDB

RDBMS_G *open source_G* pensato come alternativa a *MySQL_G*.

MariaDB ColumnStore

Database_G *column-oriented_G* basato su *MariaDB_G*, pensato per lavorare con quantità di dati molto grandi.

Media

In statistica, la media è un singolo valore numerico che descrive sinteticamente un insieme di dati. Ci sono vari tipologie di media, se non specificata, ci si riferisce alla media aritmetica, ossia:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

Modello relazionale

Modello logico di rappresentazione e strutturazione dati in un *database_G*. Esso è basato sul rappresentare i dati sotto forma di relazioni per poi manipolarli con gli operatori dell'algebra relazionale.

MongoDB

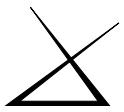
DBMS_G non relazionale, multipiattaforma e orientato ai documenti, memorizzati in formato *JSON_G*.

Multithreading

Paradigma che prevede che differenti *thread_G* vengano eseguiti contemporaneamente.

MySQL

DBMS_G relazionale, composto da un *client_G* a riga di comando e un *server_G*.



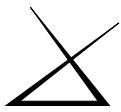
N

Node.js

È un runtime system *open source_G* multipiattaforma orientato agli eventi, che rende possibile l'I/O asincrono, per l'esecuzione di codice *JavaScript_G*.

NoSQL

Sigla di "not only SQL_G", riferito a un *database_G* indica che esso si discosta dal classico *modello relazionale_G* e che usa quindi altre forme non tabellari, come grafi o documenti, come struttura di base per i dati.



O

OLAP

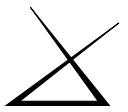
Acronimo di OnLine Analytical Processing. Indica un insieme di tecniche per l'analisi di grandi quantità di dati in poco tempo.

Open source

Software di cui l'utente finale, che può liberamente accedere al file sorgente, è in grado di modificare a suo piacimento il funzionamento, correggere eventuali errori e ridistribuire a sua volta la versione da lui elaborata.

Overfetching

Condizione che si verifica ottenendo di una quantità di dati maggiore rispetto a quella necessaria.

**P****PostgreSQL**

È un sistema di *database_G* relazionale a oggetti (*RDBMS_G*), *open source_G* e gratuito.

Progetto

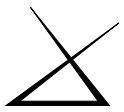
Insieme di attività che:

- devono raggiungere determinati obiettivi a partire da determinate specifiche;
- hanno una data d'inizio e una di fine fissate;
- dispongono di risorse limitate;
- consumano risorse nel loro svolgersi.

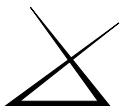
L'uscita di un progetto è un prodotto composito.

PWA

Acronimo di Progressive Web Application, particolare tipo di *web application_G* sviluppata per funzionare in un qualsiasi *browser_G* conforme allo standard *W3C_G*.

**Q****Query**

Una precisa forma di richiesta di accesso a dati, di solito salvati in un *database_G*. È solitamente associata a operazioni di lettura, scrittura, aggiornamento o eliminazione.



R

React

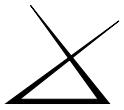
Libreria *open source_G*, *front-end_G*, *JavaScript_G* per la creazione di interfacce utente. È mantenuto da Meta e da una comunità di singoli sviluppatori e aziende.

Relational database management system (RDBMS)

DBMS_G specifico per database relazionali.

REST

Abbreviazione di REpresentational State Transfer, è uno stile architetturale basato su HTTP senza ulteriori livelli, per sistemi distribuiti stateless.



S

Selenium

Framework_G per lo sviluppo e l'esecuzione di test per servizi web, come interfacce utente ed endpoint API.

Server

Componente software che fornisce servizi e risorse ai *client_G* che lo richiedono.

Spring

Framework_G Java_G che mette a disposizione una vasta gamma di soluzioni per casistiche comuni.

Spring Boot

È una versione opinionata del *framework_G* Spring, che utilizza l'autoconfigurazione per velocizzare e facilitare lo sviluppo di *web application_G* e microservizi.

SQL

Structured Query Language è un linguaggio per *database_G* basati sul *modello relazionale_G*. È usato per la gestione di basi di dati mediante *query_G* che permettono di ottenere, modificare, cancellare, aggiungere dati.

T**Time series**

In matematica, serie di punti indicizzati temporalmente.

Thread

Linea di esecuzione. Questo termine è contestualizzato al parallelismo di esecuzione tra diversi thread appartenenti a un singolo processo.

Timescale

Timescale, o TimescaleDB, è un $DBMS_G$ relazionale ottimizzato per dati $time\ series_G$. È un'estensione di $PostgreSQL_G$.

Top-down

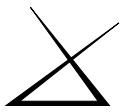
Modello di approccio alla soluzione di un problema o al compimento di un'attività, che raggiunge la soluzione partendo dall'analisi generale dell'oggetto in questione, scomponendolo nei particolari.

Transazione

Azione considerata atomica dal $DBMS_G$. Se si verifica un errore al suo interno, ogni modifica precedente causata da quella transazione viene annullata. In generale, una transazione garantisce le proprietà $ACID_G$.

TypeScript

Linguaggio di programmazione tipizzato $open\ source_G$ sviluppato da Microsoft. È un superset di $JavaScript_G$: aggiunge tipi, classi, interfacce e moduli opzionali al JavaScript tradizionale.



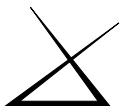
U

Underfetching

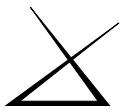
Condizione che si verifica richiedendo senza riuscire a ottenere tutte le informazioni necessarie.

UNIX epoch

Istante temporale rispetto a cui viene rappresentato il tempo nei sistemi UNIX, corrispondente alla mezzanotte del 1970-01-01.

**V****Vue**

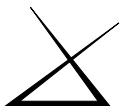
Framework_G JavaScript_G open source_G per la creazione di interfacce web e single-page applications.

**W****W3C**

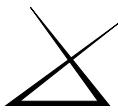
Organizzazione non governativa internazionale che definisce gli standard per lo sviluppo web.

Web application

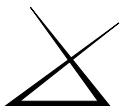
Software ospitato su un $server_G$ al quale l'utente ($client_G$) può accedervi tramite un $browser_G$.

**X****XML**

Sigla per “eXtensible Markup Language”, è un linguaggio di $markup_G$, che può fungere anche da formato per trasmettere e memorizzare dati.

**Y****YAML**

Acronimo ricorsivo per “YAML Ain’t a Markup Language”, originariamente nato come linguaggio di markup, per poi specializzarsi come formato per la memorizzazione dei dati. Oggi viene soprattutto usato come formato per i file di configurazione.



Z

Zona A

Zona della *carta di controllo*_G racchiusa tra $\bar{x} - \sigma$ e $\bar{x} + \sigma$, dove \bar{x} è la *media*_G e σ la *deviazione standard*_G dei dati. Vedere la figura 89.

Zona B

Zona della *carta di controllo*_G racchiusa tra $\bar{x} - 2\sigma$ e $\bar{x} + 2\sigma$, dove \bar{x} è la *media*_G e σ la *deviazione standard*_G dei dati. Vedere la figura 89.

Zona C

Zona della *carta di controllo*_G racchiusa tra $\bar{x} - 3\sigma$ e $\bar{x} + 3\sigma$, dove \bar{x} è la *media*_G e σ la *deviazione standard*_G dei dati. Vedere la figura 89.

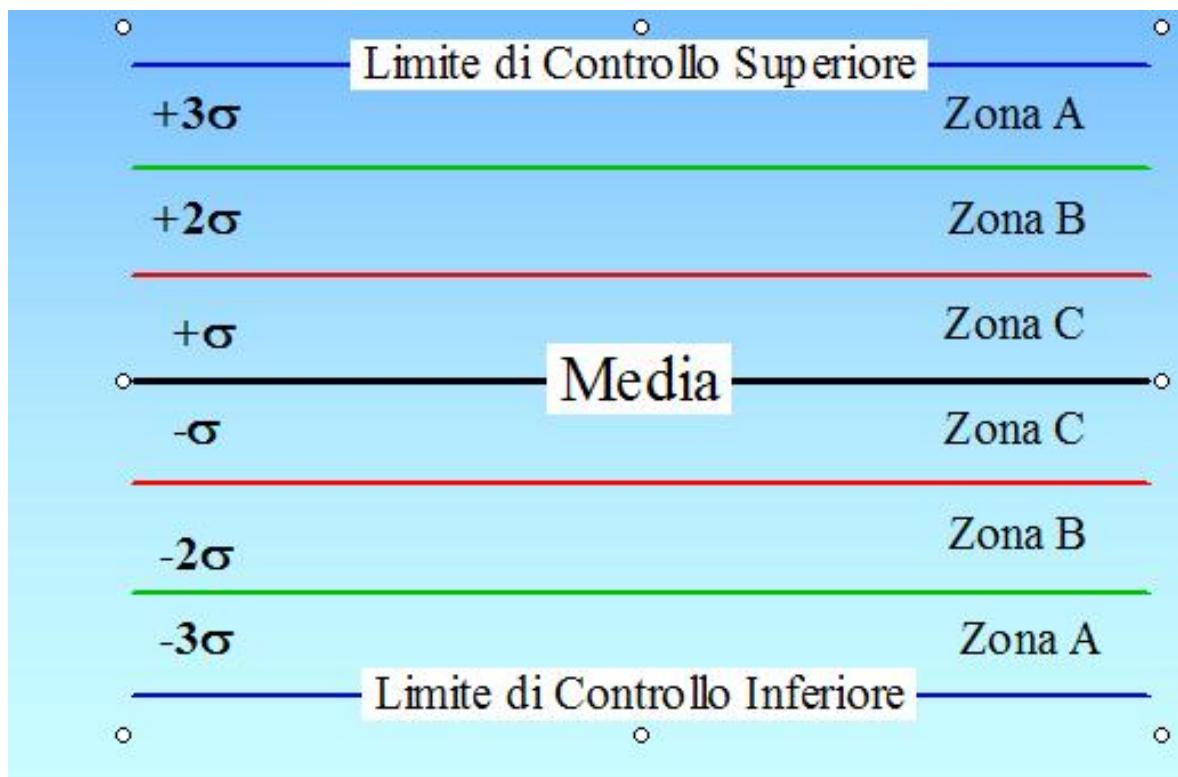


Figura 89: Zone della carta di controllo.