



AtAVi

Definizione di Prodotto v1.0.0

Sommario

Questo documento le scelte progettuali effettuate dal gruppo Co.Code per la realizzazione del *progetto_g* AtAVi.

Versione	1.0.0
Data di redazione	2017-02-02
Redazione	
Verifica	
Approvazione	
Uso	Esterno
Distribuzione	prof. Tullio Vardanega prof. Riccardo Cardin

Diario delle modifiche

Versione	Riepilogo	Autore	Ruolo	Data
----------	-----------	--------	-------	------

Indice

1	Introduzione	8
1.1	Scopo del documento	8
1.2	Scopo del prodotto	8
1.3	Glossario	8
1.4	Riferimenti	8
1.4.1	Riferimenti Normativi	8
1.4.2	Riferimenti Informativi	8
2	Standard di progetto	10
3	Architettura dell'applicazione	11
3.1	Microservizi	11
3.1.1	Virtual Assistant	11
3.1.1.1	Descrizione	11
3.1.1.2	Endpoints	11
3.1.2	Notifications	12
3.1.2.1	Descrizione	12
3.1.2.2	Endpoints	12
3.1.3	Users	13
3.1.3.1	Descrizione	13
3.1.3.2	Endpoints	13
3.1.4	Rules	14
3.1.4.1	Descrizione	14
3.1.4.2	Endpoints	15
4	Diagrammi riassuntivi dei package	18
5	Specifica dei componenti	19
6	Diagrammi di sequenza	20
7	Specifica delle Componenti	21
7.1	Back-end	21
7.1.1	Classi	21
7.1.1.1	ConversationsDAO	21
7.1.1.2	GuestsDAO	23
7.1.1.3	AdministrationWebhookService	25
7.1.1.4	AgentObservable	27
7.1.1.5	AgentObserver	28
7.1.1.6	Conversation	28
7.1.1.7	ConversationMsg	29
7.1.1.8	ConversationObservable	30
7.1.1.9	ConversationObserver	32
7.1.1.10	ConversationsDAODynamoDB	33
7.1.1.11	ConversationWebhookService	36
7.1.1.12	ErrorObservable	38
7.1.1.13	ErrorObserver	39
7.1.1.14	Guest	39
7.1.1.15	GuestObservable	41
7.1.1.16	GuestObserver	42
7.1.1.17	GuestsDAODynamoDB	44
7.1.1.18	RuleObservable	45
7.1.1.19	RuleObserver	47
7.1.1.20	SNSEvent	48

	7.1.1.21	SNSMessage	49
	7.1.1.22	SNSRecord	50
	7.1.1.23	STTParams	51
	7.1.1.24	TaskObservable	53
	7.1.1.25	TaskObserver	54
	7.1.1.26	UserObservable	55
	7.1.1.27	UserObserver	57
	7.1.1.28	VAMessageListener	58
7.2	Back-end::APIGateway		59
7.2.1	Classi		59
	7.2.1.1	Enrollment	59
	7.2.1.2	VAResponseAPIBody DA MODIFICARE DA MODIFICARE DA MODIFICARE DA MODIFICARE	60
	7.2.1.3	VocalAPI	61
	7.2.1.4	VocalLoginModuleConfig	64
7.3	Back-end::Auth		64
7.3.1	Classi		64
	7.3.1.1	UsersDAO	64
	7.3.1.2	SRUser	66
	7.3.1.3	User	66
	7.3.1.4	UsersDAODynamoDB	68
	7.3.1.5	UsersService	69
	7.3.1.6	VocalLoginModule	71
7.4	Back-end::Notifications		72
7.4.1	Classi		72
	7.4.1.1	Action	72
	7.4.1.2	Attachment	73
	7.4.1.3	ConfirmationFields	73
	7.4.1.4	NotificationChannel	74
	7.4.1.5	NotificationMessage	74
	7.4.1.6	NotificationService	75
	7.4.1.7	Purpose	76
	7.4.1.8	Topic	76
7.5	Back-end::Rules		77
7.5.1	Classi		77
	7.5.1.1	RulesDAO	77
	7.5.1.2	TasksDAO	78
	7.5.1.3	Rule	79
	7.5.1.4	RulesDAODynamoDB	81
	7.5.1.5	RulesService	82
	7.5.1.6	RuleTarget	85
	7.5.1.7	RuleTaskInstance	85
	7.5.1.8	Task	85
	7.5.1.9	TasksDAODynamoDB	86
7.6	Back-end::STT		87
7.6.1	Classi		87
	7.6.1.1	STTModule	87
	7.6.1.2	STTWatsonAdapter	87
7.7	Back-end::Utility		88
7.7.1	Classi		89
	7.7.1.1	Error	89
	7.7.1.2	LambdaContext	89
	7.7.1.3	LambdaEvent	89
	7.7.1.4	LambdaIdEvent	90
	7.7.1.5	LambdaResponse	90
	7.7.1.6	PathIdParam	90

7.7.1.7	ProcessingResult	91
7.7.1.8	StatusObject	91
7.8	Back-end::VirtualAssistant	92
7.8.1	Classi	92
7.8.1.1	AgentsDAO	92
7.8.1.2	VAModule	93
7.8.1.3	WebhookService	93
7.8.1.4	Agent	94
7.8.1.5	AgentsDAODynamoDB	95
7.8.1.6	ApiAiVAAdapter	96
7.8.1.7	ButtonObject	97
7.8.1.8	Context	97
7.8.1.9	Fulfillment	98
7.8.1.10	Metadata	98
7.8.1.11	MsgObject	99
7.8.1.12	ResponseBody	99
7.8.1.13	VAEventObject	100
7.8.1.14	VAQuery	100
7.8.1.15	VAResponse	101
7.8.1.16	VAService	101
7.9	Client	103
7.9.1	Classi	103
7.9.1.1	ConversationApp	103
7.9.1.2	ObserverAdapter	104
7.10	Client::ApplicationManager	105
7.10.1	Classi	105
7.10.1.1	ApplicationRegistryClient	105
7.10.1.2	Application	106
7.10.1.3	ApplicationManagerObserver	107
7.10.1.4	ApplicationPackage	108
7.10.1.5	ApplicationRegistryLocalClient	108
7.10.1.6	Manager	109
7.10.1.7	State	110
7.11	Client::Logic	111
7.11.1	Classi	111
7.11.1.1	DataArrivedObservable	111
7.11.1.2	DataArrivedSubject	112
7.11.1.3	HttpError	112
7.11.1.4	HttpPromise	113
7.11.1.5	Logic	113
7.11.1.6	LogicObserver	114
7.12	Client::Recorder	115
7.12.1	Classi	115
7.12.1.1	Recorder	115
7.12.1.2	RecorderConfig	116
7.12.1.3	RecorderMsg	116
7.12.1.4	RecorderWorker	117
7.12.1.5	RecorderWorkerConfig	118
7.12.1.6	RecorderWorkerMsg	119
7.12.1.7	SpeechEndObservable	119
7.12.1.8	SpeechEndSubject	119
7.13	Client::Registry	120
7.13.1	Classi	120
7.13.1.1	ApplicationLocalRegistry	120
7.14	Client::TTS	121
7.14.1	Classi	121

7.14.1.1	Player	121
7.14.1.2	PlayerObserver	122
7.14.1.3	TTSTConfig	123
7.15	Client::Utility	123
7.15.1	Classi	124
7.15.1.1	BoolObservable	124
7.15.1.2	BoolObserver	124
7.15.1.3	BoolSubject	124
7.16	RxJS	125
7.16.1	Classi	125
7.16.1.1	Observable	125
7.16.1.2	Observer	125
7.16.1.3	Subject	125
7.17	WatsonDeveloperCloud	126
7.17.1	Classi	126
7.17.1.1	SpeechToTextV1	126
8	Tracciamento	179
8.1	Tracciamento Classi-Requisiti	179
8.2	Tracciamento Componenti-Requisiti	182
A	Design Patterns	186
A.1	Architetturali	186
A.1.1	Architettura a microservizi	186
A.1.2	Architettura event-driven	186
A.1.3	Client-side discovery	186
A.1.4	Data Access Object	187
A.1.5	Dependency Injection	187
A.2	Strutturali	187
A.2.1	Facade	187
A.2.2	Adapter	188
A.3	Creazionali	188
A.3.1	Singleton	188
A.3.2	Module	188
A.4	Comportamentali	189
A.4.1	Observer	189
B	Tecnologie utilizzate	189
B.1	Promise e Observable	189
B.1.1	Promise e bluebird	189
B.1.2	Observable e RxJS v5	190
B.2	AWS SDK per JavaScript in Node.js	190
B.2.1	Node.js	190
B.2.2	AWS Lambda	190
B.2.3	Serverless Framework	190
B.2.4	DynamoDB	191
B.2.5	API.AI	191
B.2.6	JWT	191
B.2.7	Web Speech API	191
B.2.8	Speaker recognition	191
B.2.9	STT IBM Watson	191
B.2.10	Request promise	191
B.2.11	AWS Lambda	192

Elenco delle figure

1	Back-end:: ConversationsDAO	21
2	Back-end:: GuestsDAO	24
3	Back-end::AdministrationWebhookService	26
4	Back-end::AgentObservable	27
5	Back-end::AgentObserver	29
6	Back-end::Conversation	30
7	Back-end::ConversationMsg	31
8	Back-end::ConversationObservable	32
9	Back-end::ConversationObserver	33
10	Back-end::ConversationsDAODynamoDB	34
11	Back-end::ConversationWebhookService	36
12	Back-end::ErrorObservable	39
13	Back-end::ErrorObserver	40
14	Back-end::Guest	41
15	Back-end::GuestObservable	42
16	Back-end::GuestObserver	43
17	Back-end::GuestsDAODynamoDB	44
18	Back-end::RuleObservable	46
19	Back-end::RuleObserver	47
20	Back-end::SNSEvent	48
21	Back-end::SNSMessage	49
22	Back-end::SNSRecord	50
23	Back-end::STTParams	51
24	Back-end::TaskObservable	54
25	Back-end::TaskObserver	55
26	Back-end::UserObservable	56
27	Back-end::UserObserver	57
28	Back-end::VAMessageListener	58
29	Back-end::APIGateway::Enrollment	60
83	Client::ObserverAdapter	104
85	Client::ApplicationManager::Application	106
87	Client::ApplicationManager::ApplicationPackage	108
89	Client::ApplicationManager::Manager	109
91	Client::Logic::DataArrivedObservable	111
93	Client::Logic::HttpError	112
95	Client::Logic::Logic	113
97	Client::Recorder::Recorder	115
99	Client::Recorder::RecorderMsg	116
101	Client::Recorder::RecorderWorkerConfig	118
103	Client::Recorder::SpeechEndObservable	119
105	Client::Registry::ApplicationLocalRegistry	120
107	Client::TTS::PlayerObserver	122
109	Client::Utility::BoolObservable	124
111	Client::Utility::BoolSubject	124
113	RxJS::Observer	125
115	WatsonDeveloperCloud::SpeechToTextV1	126
30	Back-end::APIGateway::VAREquestAPIBody DA MODIFICARE DA MODIFICARE DA MODIFICARE DA MODIFICARE	127
31	Back-end::APIGateway::VocalAPI	128
32	Back-end::APIGateway::VocalLoginModuleConfig	129
33	Back-end::Auth:: UsersDAO	130
34	Back-end::Auth::SRUser	131
35	Back-end::Auth::User	132
36	Back-end::Auth::UsersDAODynamoDB	133

37	Back-end::Auth::UserService	134
38	Back-end::Auth::VocalLoginModule	135
39	Back-end::Notifications::Action	136
40	Back-end::Notifications::Attachment	137
41	Back-end::Notifications::ConfirmationFields	138
42	Back-end::Notifications::NotificationChannel	139
43	Back-end::Notifications::NotificationMessage	140
44	Back-end::Notifications::NotificationService	141
45	Back-end::Notifications::Purpose	142
46	Back-end::Notifications::Topic	143
47	Back-end::Rules:: RulesDAO	144
48	Back-end::Rules:: TasksDAO	145
49	Back-end::Rules::Rule	146
50	Back-end::Rules::RulesDAODynamoDB	147
51	Back-end::Rules::RulesService	148
52	Back-end::Rules::RuleTarget	149
53	Back-end::Rules::RuleTaskInstance	150
54	Back-end::Rules::Task	151
55	Back-end::Rules::TasksDAODynamoDB	152
56	Back-end::STT:: STTModule	153
57	Back-end::STT::STTWatsonAdapter	154
58	Back-end::Utility::Error	155
59	Back-end::Utility::LambdaContext	156
60	Back-end::Utility::LambdaEvent	157
61	Back-end::Utility::LambdaIdEvent	158
62	Back-end::Utility::LambdaResponse	159
63	Back-end::Utility::PathIdParam	160
64	Back-end::Utility::ProcessingResult	161
65	Back-end::Utility::StatusObject	162
66	Back-end::VirtualAssistant:: AgentsDAO	163
67	Back-end::VirtualAssistant:: VAModule	164
68	Back-end::VirtualAssistant:: WebhookService	165
69	Back-end::VirtualAssistant::Agent	166
70	Back-end::VirtualAssistant::AgentsDAODynamoDB	167
71	Back-end::VirtualAssistant::ApiAiVAAdapter	168
72	Back-end::VirtualAssistant::ButtonObject	169
73	Back-end::VirtualAssistant::Context	170
74	Back-end::VirtualAssistant::Fulfillment	171
75	Back-end::VirtualAssistant::Metadata	172
76	Back-end::VirtualAssistant::MsgObject	173
77	Back-end::VirtualAssistant::ResponseBody	174
78	Back-end::VirtualAssistant::VAEventObject	175
79	Back-end::VirtualAssistant::VAQuery	176
80	Back-end::VirtualAssistant::VAResponse	177
81	Back-end::VirtualAssistant::VAService	178

Introduzione

Scopo del documento

Lo scopo di questo documento consiste nella definizione in dettaglio della struttura e funzionamento delle componenti del progetto AtAVi. Questo documento sarà usato come guida dai *Programmatore* del gruppo.

Scopo del prodotto

Si vuole creare un'applicazione web che permetta ad un ospite, in visita all'ufficio di Zero12, di interrogare un assistente virtuale per annunciare la propria presenza, avvisare l'interessato del suo arrivo sul sistema di comunicazione aziendale (*Slack*) e nel frattempo essere intrattenuto con varie attività.

Glossario

Allo scopo di evitare ogni ambiguità nel linguaggio e rendere più semplice e chiara la comprensione dei documenti, viene allegato il “*Glossario v1.0.0*”. Le parole in esso contenute sono scritte in corsivo e marcate con una ‘g’ a pedice (p.es. *Parola_g*).

Riferimenti

Riferimenti Normativi

- “*Norme di Progetto v2.0.0*”;
- “*Analisi dei Requisiti v2.0.0*”;

Riferimenti Informativi

- Design patterns:
 - strutturali: <http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E04.pdf>;
 - creazionali: <http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E05.pdf>;
 - comportamentali: <http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E06.pdf>;
 - architetturali:
 - * <http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E08.pdf>;
 - * <http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E07.pdf>;
 - * <http://microservices.io/patterns/microservices.html>;
 - * <http://microservices.io/patterns/data/event-driven-architecture.html>;
 - * <http://microservices.io/patterns/client-side-discovery.html>;
 - * https://en.wikipedia.org/wiki/Data_access_object.
- Slide dell'insegnamento - Diagrammi delle classi: <http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E02a.pdf>;

- Slide dell'insegnamento - Diagrammi dei packages: <http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E02b.pdf>;
- Slide dell'insegnamento - Diagrammi di sequenza: <http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/E03a.pdf>;

Standard di progetto

Architettura dell'applicazione

L'architettura scelta è quella a microservizi.

Microservizi

Di seguito verranno esposti e spiegati i funzionamenti di ogni microservizio da implementare.

Virtual Assistant

Descrizione

Il microservizio Virtual Assistant fornisce le funzionalità di un assistente virtuale. Fa affidamento ad api.ai, e si occupa di inoltrare le richieste ricevute a tale infrastruttura. Avvalendosi di un database, permette di utilizzare diversi agenti durante la stessa interazione, consentendo quindi di definire diverse "applicazioni". Per ogni applicazione, si dovrà definire un agente.

Le richieste fatte all'unico endpoint di questo microservizio richiedono infatti di comunicare anche il nome dell'applicazione a cui è legata la richiesta. Questo permette di separare in diversi agenti di api.ai dialoghi legati a diverse funzionalità, consentendo lo sviluppo di diverse funzionalità da parte di sviluppatori diversi, e l'integrazione di eventuali funzionalità già esistenti senza dover modificare direttamente gli agenti di api.ai.

Per avere un completo controllo sul flusso della conversazione, si dovrà fare utilizzo di un database contenente gli agenti utilizzabili, in maniera tale che gli agenti utilizzabili siano solo quelli definiti e registrati.

Il microservizio si occupa anche di notificare, tramite l'utilizzo di AWS SNS, dell'avvenuta interazione da parte dell'utente, permettendo così il salvataggio delle conversazioni in un database di supporto, il quale potrebbe essere utilizzato magari per fini di apprendimento macchina.

Di seguito vengono esposti i vari passaggi:

- arriva una richiesta;
- interrogo il database, contenente gli agenti, utilizzando il nome dell'applicazione;
- dall'interrogazione precedente ottengo il token dell'agente relativo all'applicazione;
- invio ad api.ai il token e il testo della richiesta;
- api.ai fornisce la risposta, la quale viene "filtrata" da
`Back-end::VirtualAssistant::ApiAiVAAdapter::query(str: VAQuery): VAResponse`,
ottenendo così un formato adatto a `Back-end::VirtualAssistant::VAResponse`;
- pubblico, tramite il servizio Amazon SNS, la risposta filtrata che verrà quindi inviata al Client.

Endpoints

Ora verranno definiti gli Endpoints utilizzati per i passaggi di risorse con il microservizio Virtual Assistant.

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

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

Gli Endpoints sono:

- **/query**

- **Method:** POST;
- **Descrizione:** invia al microservizio il testo utilizzato per l'esecuzione di un'interrogazione;
- **Request:** la richiesta deve contenere i seguenti campi:

```

1 {
2     "VAQuery" : ["array di JSON contenente i dati
3                 necessari per l'interrogazione"]
3 }
```

- **Response:** la risposta deve contenere i seguenti campi:

```

1 {
2     "VAResponse" : ["array di JSON contenente dati
3                     associati alla risposta di Virtual Assistant"]
3 }
```

Notifications

Descrizione

Il microservizio Notifications si occupa di mandare messaggi di notifica nei canali adeguati per notificare gli interessati dell'arrivo di un'ospite in azienda. Fornisce le API per richiedere la lista dei possibili destinatari, e per mandare il messaggio di notifica in un determinato canale. La lista dei canali viene restituita come un'array di stringhe, ognuna delle quali rappresenta un canale.

Il microservizio si occupa di interrogare le diverse liste fornite dalla piattaforma di messaggistica scelta e di combinarle in un'unica lista. Nel nostro caso, la piattaforma di messaggistica è Slack e le diverse liste fornite riguardano utenti, canali e gruppi privati. Quando si vuole mandare un messaggio, il campo `Back-end::Notifications::NotificationMessageEvent::send_to` indica chi è il destinatario di tale messaggio.

`Back-end::Notifications::NotificationMessageEvent::msg` invece contiene il messaggio vero e proprio, nel formato definito dalla piattaforma di messaggistica su cui si appoggia il microservizio. Il formato utilizzato da Slack è consultabile qui <https://api.slack.com/docs/message-buttons>.

Endpoints

Ora verranno definiti gli Endpoints utilizzati per i passaggi di risorse con il microservizio Notifications.

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

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

Gli Endpoints sono:

- **/**
 - **Method:** GET;
 - **Descrizione:** restituisce la lista dei possibili destinatari;

- **Response:** la risposta deve contenere i seguenti campi:

```
1 {  
2     receiver_array : ["array di stringhe dei possibili  
3         destinatari"]  
3 }
```

- **Method:** POST;
- **Descrizione:** invia la notifica ad una determinata persona;
- **Request:** la richiesta deve contenere i seguenti campi:

```
1 {  
2     rule : ["array di JSON contenente i dati associati  
3         alla direttiva da inviare"]  
3     sender : ["stringa contenente il mittente della  
4         notifica "]  
4     receiver : ["stringa contenente il destinatario  
5         della notifica "]  
5     msg : ["stringa contenente il messaggio da inviare"]  
6 }
```

Users

Descrizione

Il microservizio Users si occupa della gestione degli amministratori del nostro sistema. Esso fornisce delle API REST per modificare i dati relativi agli amministratori del nostro sistema presenti in un database. Viene integrato col servizio di Speech Recognition dall'API Gateway, per fornire la possibilità di effettuare il login, tramite impronta vocale, nel sistema.

Endpoints

Ora verranno definiti gli Endpoints utilizzati per i passaggi di risorse con il microservizio Users. Per ogni risorsa sono stati specificati i formati per lo scambio dei dati in JSON:

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

Gli Endpoints sono:

- **/auth/users**

- **Method:** GET;
- **Descrizione:** restituisce la lista degli Users;
- **Response:** la risposta deve contenere i seguenti campi:

```
1 {  
2  
3 }
```

- **Method:** POST;
- **Descrizione:** vengono inviati i dati necessari alla registrazione;

- **Request:** la richiesta deve contenere i seguenti campi:

```

1 {
2     user_array : ["array di stringhe degli utenti
3                 esistenti"]
3 }
```

- **/auth/users/:username**

- **Method:** PUT;
- **Descrizione:** vengono modificati i dati dell'utente tramite sovrascrittura;
- **Request:** la richiesta deve contenere i seguenti campi:

```

1 {
2     user : ["array di JSON contenente i nuovi dati da
3           inserire"]
3 }
```

- **Method:** DELETE;
- **Descrizione:** viene eliminato un utente;
- **Request:** la richiesta deve contenere i seguenti campi:

```

1 {
2     username : ["stringa contenente l'username dell'
3               utente da eliminare"]
3 }
```

- **Method:** GET;
- **Descrizione:** vengono ricevuti i dati relativi ad un utente;
- **Request:** la richiesta deve contenere i seguenti campi:

```

1 {
2     username : ["stringa contenente l'username dell'
3               utente desiderato"]
3 }
```

- **Response:** la risposta deve contenere i seguenti campi:

```

1 {
2     user : ["array di JSON contenente l'utente
3           desiderato"]
3 }
```

Rules

Descrizione

Il microservizio Rules si occupa della gestione delle direttive del sistema. Una direttiva è un'istruzione che viene data da un amministratore al sistema, la quale permette di modificare il comportamento del sistema stesso al verificarsi di certe condizioni. Tali condizioni possono essere legate alla persona che interagisce col sistema, la sua azienda di provenienza, oppure alla persona desiderata che viene richiesta.

Il sistema fornisce una serie di funzioni per modificare il suo comportamento, le quali indicano il modo in cui esso debba essere cambiato.

Una direttiva è costituita da:

- una lista di target, che indica gli obiettivi ai quali deve essere applicata la direttiva;
- un'istanza di funzione, che indica quale delle funzioni disponibili deve essere applicata e, nel caso in cui tale funzione abbia dei parametri modificabili, con quali valori di quest'ultimi deve essere chiamata;
- un nome, il quale permette agli amministratori di identificare le diverse direttive;
- un id, il quale identifica univocamente la funzione all'interno del sistema;
- una flag di abilitazione, che permette di abilitare e disabilitare l'applicazione della direttiva da parte del sistema.

Endpoints

Ora verranno definiti gli Endpoints utilizzati per i passaggi di risorse con il microservizio Rules. Per ogni risorsa sono stati specificati i formati per lo scambio dei dati in JSON:

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

Gli Endpoints sono:

- **/impostazioni**

- **Method:** GET;
- **Descrizione:** viene ricevuta la lista delle direttive;
- **Response:** la risposta deve contenere i seguenti campi:

```
1 {  
2     rule_array : ["array di stringhe contenente la lista  
3                 delle direttive"]  
}
```

- **Method:** POST;
- **Descrizione:** viene creata una nuova direttiva
- **Request:** la richiesta deve contenere i seguenti campi:

```
1 {  
2     rule : ["array di JSON contenente i dati associati  
3           alla direttiva da creare"]  
}
```

- **/impostazioni/:id**

- **Method:** PUT;
- **Descrizione:** viene modificata una direttiva tramite sovrascrittura;
- **Request:** la richiesta deve contenere i seguenti campi:


```
1 {  
2     rule : ["array di JSON contenente i nuovi dati  
           associati alla direttiva da modificare"]  
3 }
```

– **Method:** GET;

– **Descrizione:** vengono richiesti dati relativi ad una specifica direttiva;

– **Request:** la richiesta deve contenere i seguenti campi:

```
1 {  
2     id_rule : ["stringa contenente l'id della direttiva  
              desiderata"]  
3 }
```

– **Response:** la risposta deve contenere i seguenti campi:

```
1 {  
2     rule : ["array di JSON contenente i dati associati  
           alla direttiva desiderata"]  
3 }
```

– **Method:** DELETE;

– **Descrizione:** viene eliminata una direttiva;

– **Request:** la richiesta deve contenere i seguenti campi:

```
1 {  
2     id_rule : ["stringa contenente l'id della direttiva  
              da eliminare"]  
3 }
```

- /impostazioni/functions/

– **Method:** GET;

– **Descrizione:** viene richiesta la lista dei tipi di funzioni presenti nel sistema;

– **Response:** la risposta deve contenere i seguenti campi:

```
1 {  
2     function_array : ["array di stringhe contenente la  
                      lista dei tipi di funzioni presenti nel sistema"]  
3 }
```

– **Method:** POST;

– **Descrizione:** viene creato un nuovo tipo di funzione;

– **Request:** la richiesta deve contenere i seguenti campi:

```
1 {  
2     function : ["array di JSON contenente il tipo di  
               funzione da creare"]  
3 }
```

- /impostazioni/functions/:type

- **Method:** GET;
- **Descrizione:** viene richiesta la descrizione di una funzione;
- **Request:** la richiesta deve contenere i seguenti campi:

```
1 {  
2     id_function : ["stringa contenente l'id della  
3                 funzione richiesta"]  
}
```

- **Response:** la risposta deve contenere i seguenti campi:

```
1 {  
2     description_function : ["stringa contenente la  
3                             descrizione della funzione richiesta"]  
}
```

- **Method:** PUT;
- **Descrizione:** viene modificata una funzione tramite sovrascrittura;
- **Request:** la richiesta deve contenere i seguenti campi:

```
1 {  
2     function : ["array di JSON contenente i nuovi dati  
3               associati alla funzione da modificare"]  
}
```

Diagrammi riassuntivi dei package

Specifica dei componenti

Diagrammi di sequenza

Specifica delle Componenti

Back-end

Package contenente tutte le componenti che costituiscono il back-end. Le componenti sono organizzate secondo il pattern a microservizi.

Classi

ConversationsDAO

ConversationsDAO.png ConversationsDAO.png ConversationsDAO.png

images/Class ConversationsDAO.png

Figura 1: Back-end:: ConversationsDAO

- **Nome:** ConversationsDAO;
- **Tipo:** Interface;

- **Descrizione:** questa classe si occupa di astrarre le modalità d'interazione al database contenente le conversazioni;
- **Utilizzo:** fornisce a `ConversationWebhookService` un meccanismo per accedere al database contenente le conversazioni, senza conoscerne le modalità di implementazioni e di persistenza di quest'ultimo. Permette operazioni di lettura, scrittura e rimozione di utenti registrati;
- **Attributi:**
 - `db: AWS::DynamoDB`
Attributo che permette di contattare il database contenente le conversazioni;
 - `guest_id: String`
Attributo contenente l'id dell'ospite identificato;
- **Metodi:**
 - + `getConversationList(): ConversationObservable`
L'`Observable` restituito manderà agli `Observer` le conversazioni ottenute, uno alla volta, e poi chiama il loro metodo `complete`. Nel caso in cui si verifichi un errore, gli `Observer` iscritti verranno notificati tramite la chiamata del loro metodo `error` con i dati relativi all'errore verificatosi;
 - + `getConversation(session_id: String, timestamp: String): ConversationObservable`
Metodo che permette di ottenere una conversazione a partire dall'identificativo della sessione e dal suo timestamp.
L'`Observable` restituito riceverà l'oggetto rappresentante tale `Conversation`, e verrà completato. Nel caso in cui la conversazione richiesta non sia presente nel database, gli `Observer` interessati non riceveranno alcun valore, ma verranno notificati tramite la chiamata del loro metodo `error()`;
Parametri:
 - * `session_id: String`
Parametro contenente l'id della sessione della conversazione da ricevere;
 - * `timestamp: String`
Parametro contenente il timestamp relativo all'inizio della conversazione;
 - + `addConversation(conversation: Conversation): ErrorObservable`
Metodo che permette di aggiungere una conversazione al database. L'`Observable` restituito non riceverà alcun valore, ma verrà completato in caso di aggiunta della conversazione avvenuta con successo. In caso di errore durante l'aggiunta della conversazione, gli `Observer` interessati verranno notificati tramite la chiamata del loro metodo `error()` con i dati relativi all'errore verificatosi;
Parametri:
 - * `conversation: Conversation`
Parametro contenente la conversazione da inserire;
 - + `addMessage(msg: ConversationMessage, timestamp: String, session_id: String): ErrorObservable`
Metodo che permette di aggiungere un messaggio relativo ad una conversazione al database a partire da un messaggio, un timestamp relativo e l'identificativo della relativa conversazione. L'`Observable` restituito non riceverà alcun valore, ma verrà completato in caso di aggiunta del messaggio avvenuta con successo. In caso di errore durante l'aggiunta del messaggio, gli `Observer` interessati verranno notificati tramite la chiamata del loro metodo `error()` con i dati relativi all'errore verificatosi;
Parametri:
 - * `msg: ConversationMessage`
Parametro contenente il messaggio;

```

* timestamp: String
    Parametro contenente il timestamp relativo all'inizio della conversazione;

* session_id: String
    Parametro contenente l'id della sessione della conversazione alla quale aggiungere
    il messaggio;

+ removeConversation(session_id: String, timestamp: String): ErrorObservable
    Metodo che permette di rimuovere una conversazione a partire dall'identificativo della
    sessione e dal suo timestamp. L'Observable restituito non riceverà alcun valore, ma
    verrà completato in caso di aggiunta della conversazione avvenuta con successo. In ca-
    so di errore durante l'aggiunta della conversazione, gli Observer interessati verranno
    notificati tramite la chiamata del loro metodo error() con i dati relativi all'errore ve-
    rificatosi;
    Parametri:

    * session_id: String
        Parametro contenente l'id della sessione della conversazione da eliminare;

    * timestamp: String
        Parametro contenente il timestamp del messaggio da eliminare;

```

GuestsDAO

- **Nome:** GuestsDAO;
- **Tipo:** Interface;
- **Descrizione:** questa interfaccia si occupa di astrarre le modalità d'interazione al database contenente gli ospiti conosciuti;
- **Utilizzo:** fornisce i metodi per accedere al database contenente i dati relativi agli ospiti conosciuti, senza conoscerne le modalità di implementazioni e di persistenza di quest'ultimo. Permette operazioni di lettura, scrittura e rimozione di ospiti conosciuti;
- **Attributi:**
 - db: AWS::DynamoDB
Attributo contenente;
- **Metodi:**

```

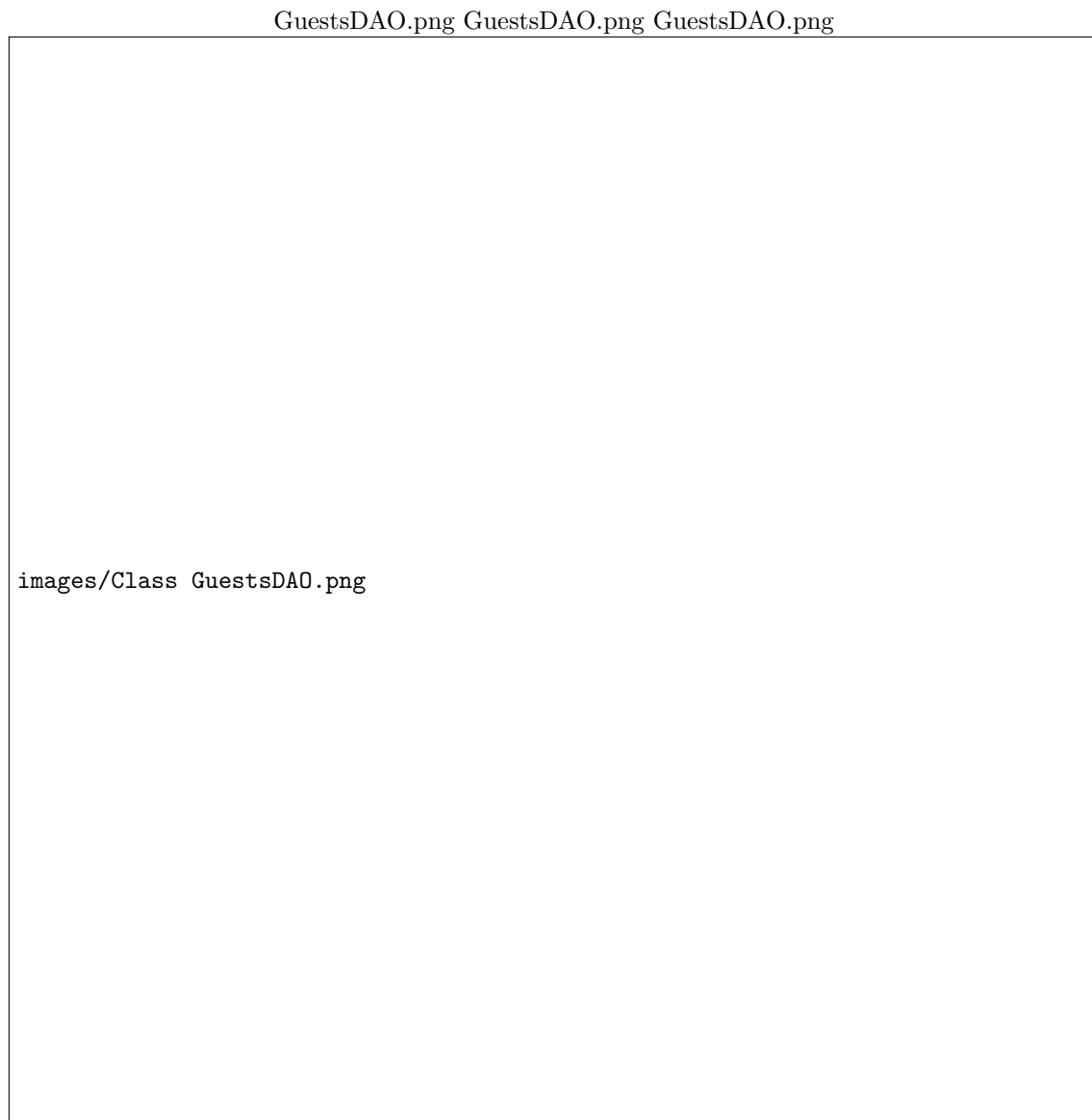
+ getGuestList(): GuestObservable
    L'Observable restituito manderà agli Observer gli ospiti ottenuti, uno alla volta, e poi
    chiama il loro metodo complete. Nel caso in cui si verifichi un errore, gli Observer
    iscritti verranno notificati tramite la chiamate del loro metodo error con i dati relativi
    all'errore verificatosi;

+ getGuest(name: String, company: String): GuestObservable
    Metodo che permette di ottenere un ospite a partire dal nome e l'azienda di provenienza.
    L'Observable restituito riceverà l'oggetto rappresentante tale Guest, e verrà comple-
    tato. Nel caso in cui l'ospite richiesto non sia presente nel database, gli Observer
    interessati non riceveranno alcun valore, ma verranno notificati tramite la chiamata del
    loro metodo error();
    Parametri:

    * name: String
        Parametro contenente il nome dell'ospite;

    * company: String
        Parametro contenente l'azienda di provenienza dell'ospite;

```


**Figura 2:** Back-end:: GuestsDAO

+ updateGuest(guest: Guest): ErrorObservable

Metodo che permette di aggiornare un ospite. L'**Observable** restituito non riceverà alcun valore, ma verrà completato in caso di aggiunta dell'ospite avvenuta con successo. In caso di errore durante l'aggiunta dell'ospite, gli **Observer** interessati verranno notificati tramite la chiamata del loro metodo **error()** con i dati relativi all'errore verificatosi;

Parametri:

* **guest: Guest**

Parametro contenente l'ospite da aggiornare;

+ removeGuest(name: String, company: String): ErrorObservable

Metodo che permette di eliminare un ospite dal database, a partire dal suo nome e azienda di provenienza. L'**Observable** restituito non riceverà alcun valore, ma verrà completato in caso di eliminazione dell'ospite avvenuta con successo. In caso di errore durante l'eliminazione dell'ospite, gli **Observer** interessati verranno notificati tramite la chiamata del loro metodo **error()** con i dati relativi all'errore verificatosi;

Parametri:

```

* name: String
    Parametro contenente il nome dell'ospite;

* company: String
    Parametro contenente l'azienda di provenienza dell'ospite;

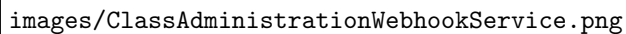
+ addGuest(guest: Guest): ErrorObservable
    Metodo che permette di aggiungere un ospite al database. L'Observable restituito
    non riceverà alcun valore, ma verrà completato in caso di aggiunta dell'ospite avvenuta
    con successo. In caso di errore durante l'aggiunta dell'ospite, gli Observer interessati
    verranno notificati tramite la chiamata del loro metodo error() con i dati relativi
    all'errore verificatosi;
    Parametri:

    * guest: Guest
        Parametro contenente l'ospite da aggiungere;

```

AdministrationWebhookService

- **Nome:** AdministrationWebhookService;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di implementare l'interfaccia WebhookService, realizzando un Webhook che fornisce una risposta all'Agent di amministrazione;
- **Utilizzo:** fornisce il metodo che si occupa di rispondere alle chiamate al microservizio da parte dell'Agent di amministrazione;
- **Padre:** <<interface>> WebhookService;
- **Attributi:**
 - rules: RulesDAO
Attributo che permette di contattare RulesDAO, il quale fornisce i meccanismi di accesso al database contenente le Rule;
 - users: UsersDAO
Attributo che permette di contattare UsersDAO, il quale fornisce i meccanismi di accesso al database contenente gli utenti registrati;
- **Metodi:**
 - checkToken(token: String): boolean
Metodo che permette di controllare l'autenticità di un JWT. Utilizzato dal metodo webhook per controllare che la richiesta sia stata fatta da un amministratore autenticato prima di compiere qualsiasi altra azione;
Parametri:
 - * token: String
Parametro contenente il JWT per l'autenticazione;
 - + webhook(event: LambdaEvent, context: LambdaContext): void
Questo metodo soddisfa i requisiti per webhook descritti da api.ai. Viene chiamato ad ogni interazione dell'agente di amministrazione, e per prima cosa si occupa di verificare che nella richiesta sia presente un JSON Web Token (JWT) che confermi un'autenticazione avvenuta con successo. In caso di mancata autenticazione, lo stato della risposta sarà impostato a 403. Nel caso in cui il token sia presente e valido (la firma sia valida ed il token non sia scaduto), si occupa di eseguire l'operazione richiesta dall'assistente virtuale. Tale operazione è specificata in fulfillment.messages (vedi classi ProcessingResult, Fullfillment e MsgObject per eventuali chiarimenti riguardanti il formato della richiesta fatta da api.ai al microservizio). In particolare, all'interno di



images/ClassAdministrationWebhookService.png

Figura 3: Back-end::AdministrationWebhookService

`messages` sarà presente un messaggio con `type=4`, il quale relativo attributo `payload` consisterà in un oggetto di tipo `WebhookCmd`. In base al campo `cmd` di tale oggetto questo metodo interpreterà il formato del campo `params`, ricavando i parametri necessari ad eseguire il comando specificato e poi eseguendo l'azione richiesta. In caso di successo verrà richiesto all'utente se vuole compiere qualche altra azione utilizzando il campo `speech` dell'oggetto `WebhookResponse` utilizzato per rispondere. In caso di mancanza del comando (nessun messaggio con `type=4` in `fulfillment.messages`), il campo `speech` della risposta sarà copiato da `fulfillment.speech` della richiesta, risultando quindi in una risposta uguale a quella fornita da `api.ai` in assenza di `webhook`;

Parametri:

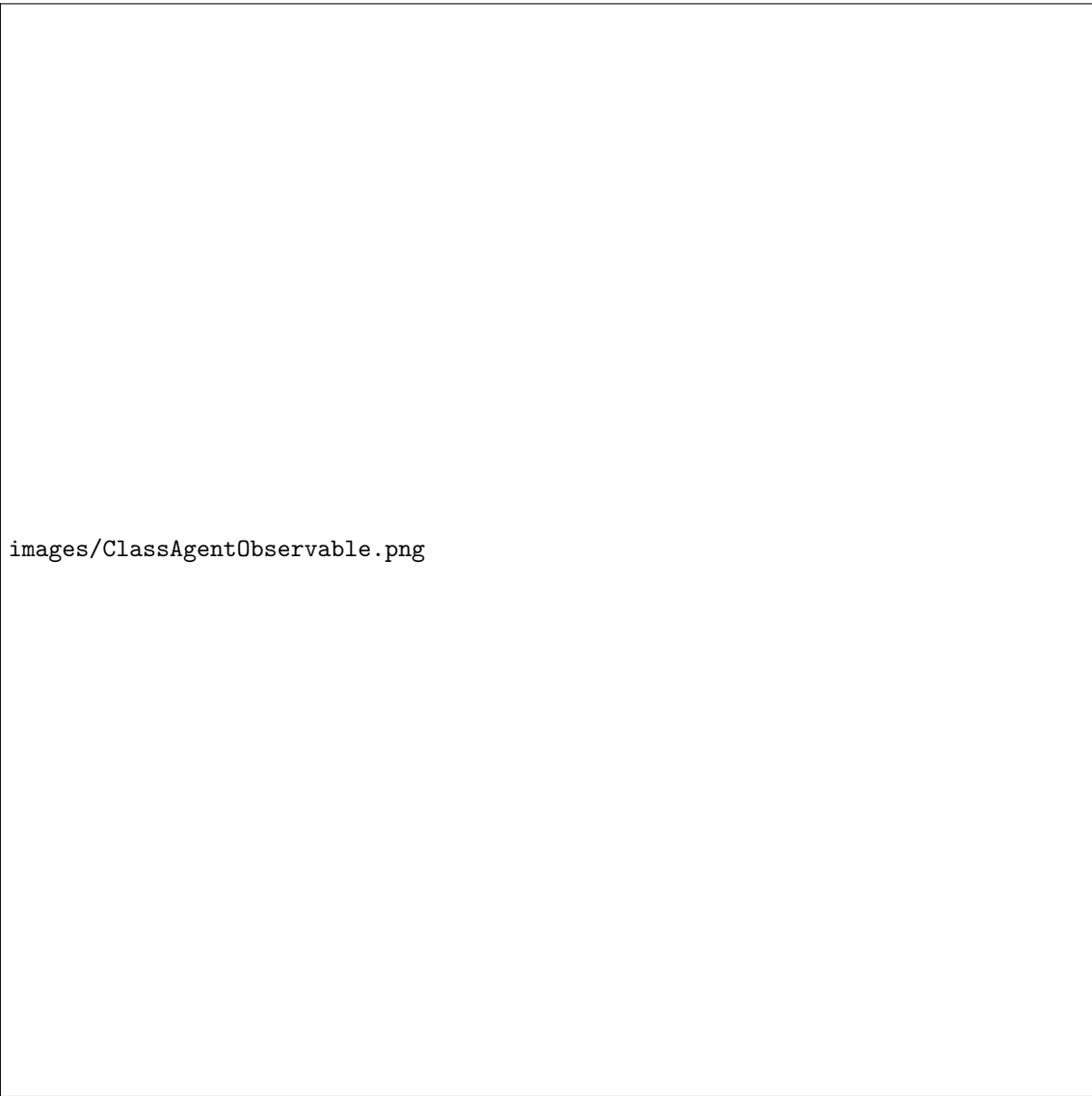
```
* event: LambdaEvent
    Parametro contenente;

* context: LambdaContext
    Parametro contenente;

+ <<Create>> createAdministrationWebhookService(rulesdao: RulesDAO, usersdao:
```

```
UsersDAO): AdministrationWebhookService  
zzz;  
Parametri:  
* rulesdao: RulesDAO  
  Attributo contenente il RulesDAO;  
* usersdao: UsersDAO  
  Attributo contenente lo UsersDAO;
```

AgentObservable



images/ClassAgentObservable.png

Figura 4: Back-end::AgentObservable

- **Nome:** AgentObservable;
- **Tipo:** Class;
- **Descrizione:** questa classe implementa un Observable che permette l'iscrizione di AgentObserver;

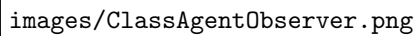
- **Utilizzo:** fornisce i meccanismi necessari per il passaggio di una serie di **Agent** ad un **Observer** interessato;
- **Padre:** **Observable**;
- **Metodi:**
 - + `<<Create>> createAgentObservable(onSubscription: function(observer: AgentObserver) : void): AgentObservable`
 Constructor di **AgentObservable**;
 Parametri:
 - * `onSubscription: function(observer: AgentObserver) : void`
 Funzione che verrà eseguita quando un **Observer** si iscrive all'**Observable**. Si occupa di passare i dati all'**Observer**, chiamando il metodo `next(agent: Agent)`. Quando non ci sono più dati da restituire, si occupa di chiamare il metodo `complete()`. Nel caso in cui si verificasse un errore, si occupa di chiamare il metodo `error(err: Object)` con i dati relativi all'errore verificatosi;
 - + `subscribe(observer: AgentObserver): Subscription`
 Metodo che permette ad uno **AgentObserver** interessato di iscriversi a questo **Observable**;
 Parametri:
 - * `observer: AgentObserver`
 Observer che si vuole iscrivere;

AgentObserver

- **Nome:** **AgentObserver**;
- **Tipo:** **Class**;
- **Descrizione:** classe che rappresenta un **Observer** che si aspetta dati di tipo **Agent**;
- **Utilizzo:** implementa il metodo `next()` dell'interfaccia, in maniera tale che accetti dati di tipo **Agent**;
- **Metodi:**
 - + `next(agent: Agent): void`
 Metodo che permette agli **Observable** di notificare l'**Observer** con dati di tipo **Agent**. Definisce inoltre le operazioni che l'**Observer** compierà all'arrivo di tali dati;
 Parametri:
 - * `agent: Agent`
 Parametro contenente l'**Agent** mandato dall'**Observable**;

Conversation

- **Nome:** **Conversation**;
- **Tipo:** **Class**;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare gli attributi relativi ad una conversazione, la quale dovrà essere salvata nel database;
- **Utilizzo:** fornisce gli attributi relativi ad una conversazione;
- **Attributi:**
 - + `messages: ConversationMsgArray`
 Attributo contenente l'array dei messaggi contenuti in una conversazione;



images/ClassAgentObserver.png

Figura 5: Back-end::AgentObserver

+ **session_id:** String

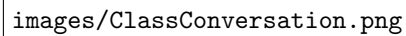
Attributo contenente l'id della sessione relativa alla conversazione;

+ **timestamp:** String

Attributo contenente il timestamp relativo alla conversazione. Tutti i messaggi di una conversazione hanno lo stesso timestamp, il quale indica l'inizio della conversazione;

ConversationMsg

- **Nome:** ConversationMsg;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare gli attributi relativi ad un messaggio all'interno di una conversazione;
- **Utilizzo:** fornisce gli attributi relativi un messaggio. La classe **Conversation** contiene un array di essi, in modo da costruire l'intera conversazione;



images/ClassConversation.png

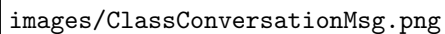
Figura 6: Back-end::Conversation

- **Attributi:**

- + **text:** `String`
Attributo contenente il testo del messaggio;
- + **sender:** `String`
Attributo contenente il mittente del messaggio;
- + **timestamp:** `String`
Attributo contenente il timestamp del messaggio;

ConversationObservable

- **Nome:** `ConversationObservable`;
- **Tipo:** `Class`;
- **Descrizione:** questa classe implementa un `Observable` che permette l'iscrizione di `ConversationObserver`;



images/ClassConversationMsg.png

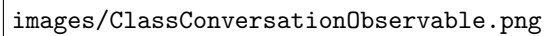
Figura 7: Back-end::ConversationMsg

- **Utilizzo:** fornisce i meccanismi necessari per il passaggio di una serie di `Conversation` ad un `Observer` interessato;
- **Padre:** `Observable`;
- **Metodi:**

```
+ <<Create>> createConversationObservable(onSubscription: function(observer:
ConversationObserver) : void): ConversationObservable
Constructor di ConversationObservable;
```

Parametri:

```
* onSubscription: function(observer: ConversationObserver) : void
Funzione che verrà eseguita quando un Observer si iscrive all'Observable. Si occupa di passare i dati all'Observer, chiamando il metodo next(conversation: Conversation). Quando non ci sono più dati da restituire, si occupa di chiamare il metodo complete(). Nel caso in cui si verificasse un errore, si occupa di chiamare il metodo error(err: Object) con i dati relativi all'errore verificatosi;
```

images/ClassConversationObservable.png

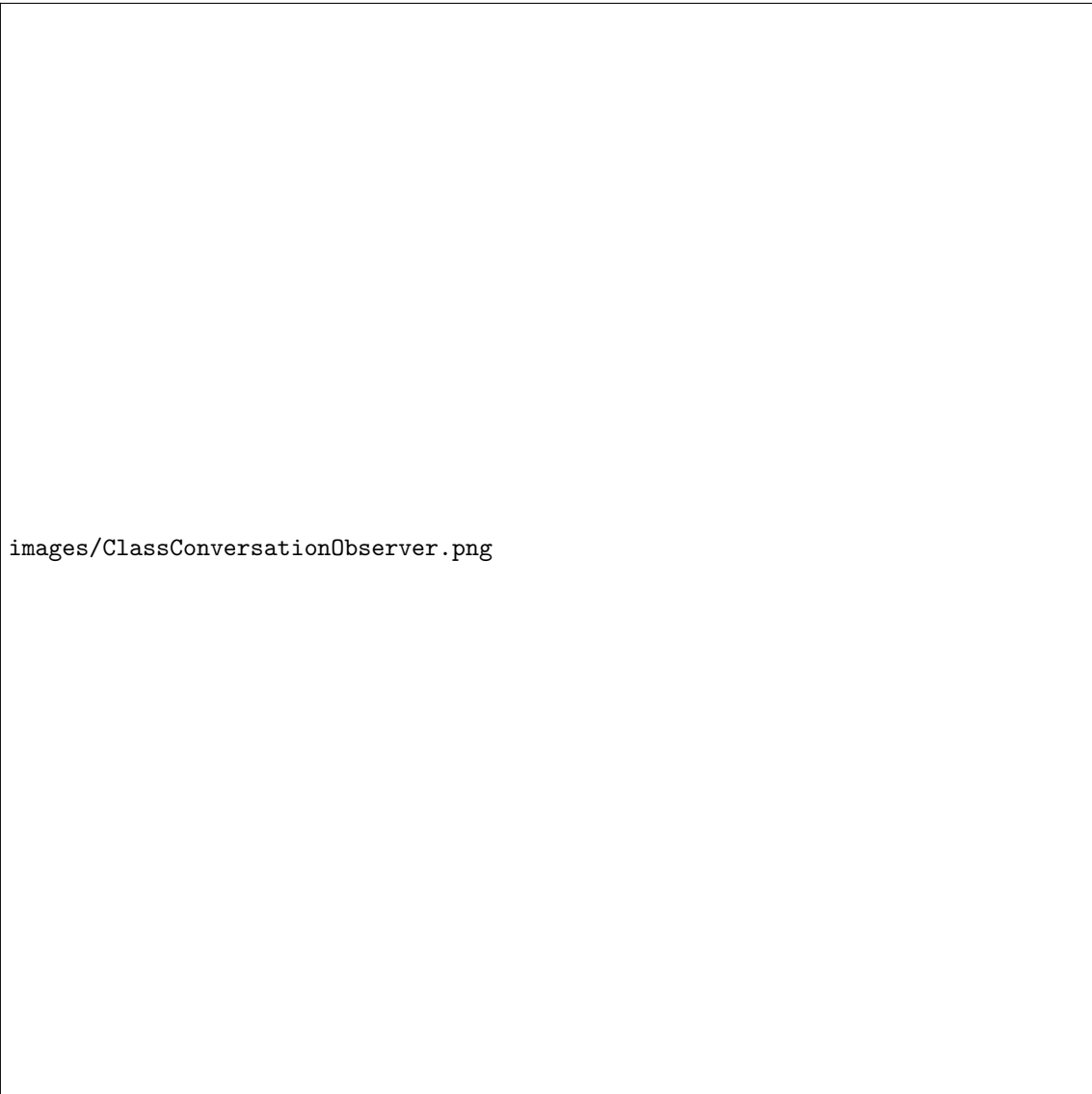
Figura 8: Back-end::ConversationObservable

```
+ subscribe(observer: ConversationObserver): Subscription
Metodo che permette ad uno ConversationObserver interessato di iscriversi a questo
Observable;
Parametri:

* observer: ConversationObserver
  Observer che si vuole iscrivere;
```

ConversationObserver

- **Nome:** ConversationObserver;
- **Tipo:** Class;
- **Descrizione:** classe che rappresenta un Observer che si aspetta dati di tipo Conversation.
;



images/ClassConversationObserver.png

Figura 9: Back-end::ConversationObserver

- **Utilizzo:** implementa il metodo `next()` dell'interfaccia, in maniera tale che accetti dati di tipo `Conversation`;

- **Metodi:**

```
+ next(conversation: Conversation): void
```

Metodo che permette agli `Observable` di notificare l'`Observer` con dati di tipo `Conversation`.
Definisce inoltre le operazioni che l'`Observer` compierà all'arrivo di tali dati;

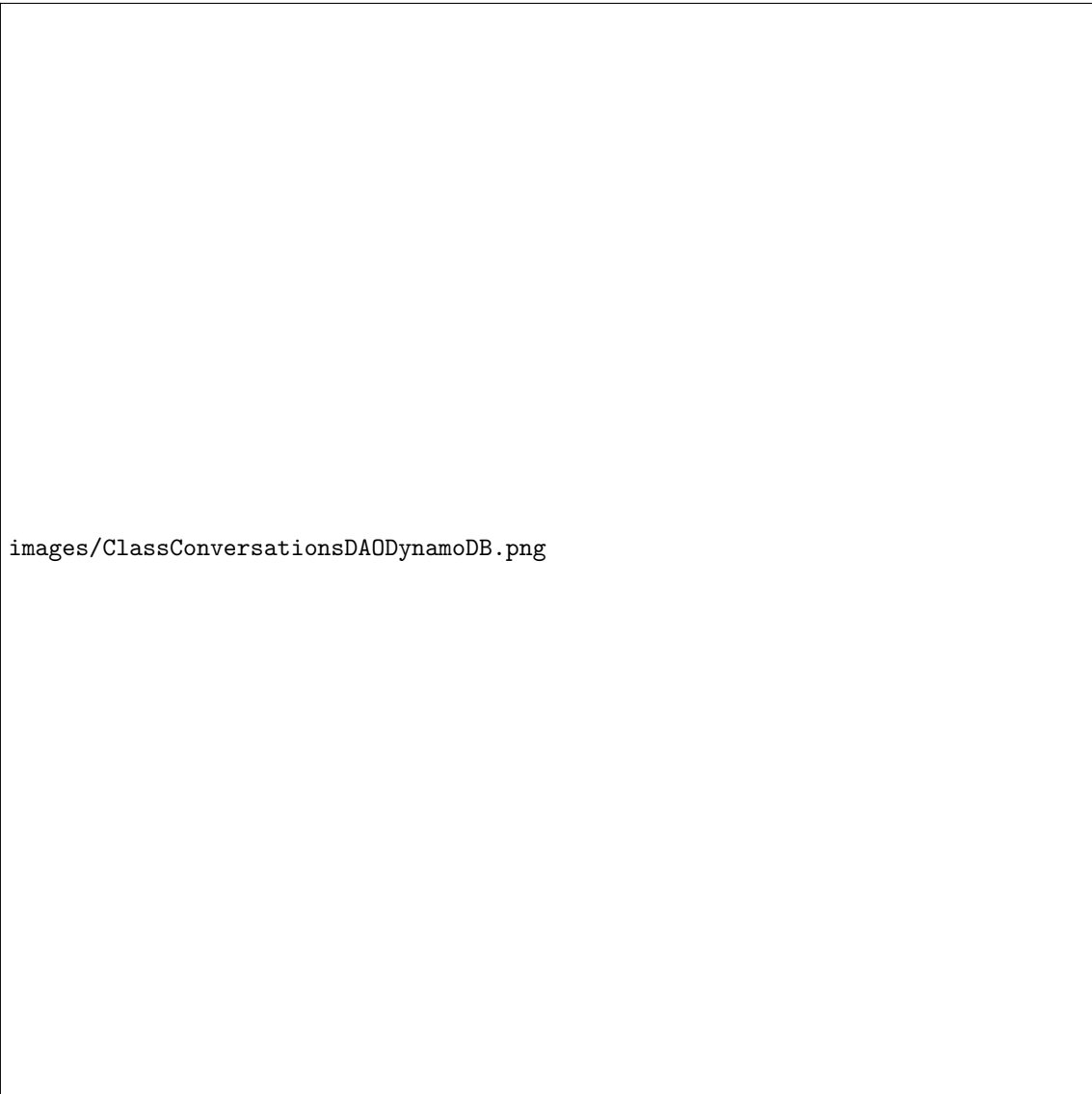
Parametri:

```
* conversation: Conversation
```

Parametro contenente la `Conversation` mandata dall'`Observable`;

ConversationsDAODynamoDB

- **Nome:** `ConversationsDAODynamoDB`;
- **Tipo:** `Class`;



images/ClassConversationsDAODynamoDB.png

Figura 10: Back-end::ConversationsDAODynamoDB

- **Descrizione:** classe che si occupa di implementare l'interfaccia `ConversationsDAO`, utilizzando un database `DynamoDB` come supporto per la memorizzazione dei dati;
- **Utilizzo:** implementa i metodi dell'interfaccia `ConversationsDAO` interrogando un database `DynamoDB`. Utilizza `AWS::DynamoDB::DocumentClient` per l'accesso al database. La dependency injection dell'oggetto `AWS::DynamoDB` viene fatta utilizzando il costruttore;
- **Attributi:**
 - `db: AWS::DynamoDB`
Attributo contenente un riferimento al modulo di Node.js utilizzato per l'accesso al database `DynamoDB` contenente la tabella degli utenti;
- **Metodi:**
 - + `addConversation(conversation: Conversation): ErrorObservable`
Implementazione del metodo definito nell'interfaccia `ConversationsDAO`. Utilizza il metodo `put` del `DocumentClient` per aggiungere la conversazione al database;
Parametri:

```
* conversation: Conversation
    Parametro contenente la conversazione da inserire;
```

```
+ getConversation(session_id: String, timestamp: String): ConversationObservable
Implementazione del metodo definito nell'interfaccia ConversationsDAO. Utilizza il me-
todo get del DocumentClient per ottenere i dati relativi ad una Conversation dal
database;
Parametri:
```

```
* session_id: String
    Parametro contenente l'id della sessione della conversazione da ricevere;
```

```
* timestamp: String
    Parametro contenente il timestamp relativo all'inizio della conversazione;
```

```
+ getConversationList(): ConversationObservable
Implementazione del metodo dell'interfaccia ConversationsDAO. Utilizza il metodo scan
del DocumentClient per ottenere la lista delle conversazioni dal database;
```

```
+ removeConversation(session_id: String, timestamp: String): ErrorObservable
Implementazione del metodo dell'interfaccia ConversationsDAO. Utilizza il metodo de-
lete del DocumentClient per eliminare una conversazione dal database;
Parametri:
```

```
* session_id: String
    Parametro contenente l'id della sessione della conversazione da eliminare;
```

```
* timestamp: String
    Parametro contenente il timestamp relativo all'inizio della conversazione;
```

```
+ <> createConversationsDAODynamoDB(guest_id: String, db: AWS::DynamoDB):
ConversationsDAODynamoDB
Constructor della classe ConversationsDAODynamoDB. Permette di effettuare la depen-
dency injection di AWS::DynamoDB;
Parametri:
```

```
* guest_id: String
    sdadnoadnaosd;
```

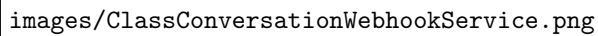
```
* db: AWS::DynamoDB
    Parametro contenente un riferimento al modulo di Node.js da utilizzare per l'accesso
    al database DynamoDB contenente la tabella delle conversazioni;
```

```
+ addMessage(msg: ConversationMsg, timestamp: String, session_id: String):
ErrorObservable
Implementazione del metodo definito nell'interfaccia ConversationsDAO. Utilizza il me-
todo put del DocumentClient per aggiungere un messaggio ad una conversazione;
Parametri:
```

```
* msg: ConversationMsg
    Parametro contenente il messaggio. ;
```

```
* timestamp: String
    Parametro contenente il timestamp relativo all'inizio della conversazione;
```

```
* session_id: String
    Parametro contenente l'id della sessione della conversazione dove aggiungere il
    messaggio;
```



images/ClassConversationWebhookService.png

Figura 11: Back-end::ConversationWebhookService

ConversationWebhookService

- **Nome:** ConversationWebhookService;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di implementare l'interfaccia **WebhookService**, implementando un Webhook che fornisce una risposta ad api.ai;
- **Utilizzo:** fornisce il metodo che si occupa di rispondere alle chiamate al microservizio da parte di api.ai;
- **Padre:** <<interface>> WebhookService;
- **Attributi:**
 - **users:** UsersDAO
Attributo che permette di contattare UsersDAO, il quale fornisce i meccanismi di accesso al database contenente gli utenti registrati;

- **guests:** **GuestsDAO**

Attributo che permette di contattare GuestDAO, il quale fornisce i meccanismi di accesso al database contenente gli ospiti;

- **conversation:** **ConversationsDAO**

Attributo che permette di contattare ConversationDAO, il quale fornisce i meccanismi di accesso al database delle conversazioni;

• Metodi:

+ **webhook(event: LambaEvent, context: LambdaContext): void**

Questo metodo soddisfa i requisiti per webhook descritti da api.ai. Si occupa di cercare nei database la presenza di interazioni passate con la persona con cui sta avvenendo l'interazione, e di verificare se la persona in questione può essere un amministratore del sistema. Nel caso di interazioni passate il context accoglienza viene riempito con azienda e dati della persona con cui l'ospite ha avuto il maggior numero di incontri in passato, e viene chiesto di confermare se la persona indicata è quella a cui l'ospite è effettivamente interessato. Nel caso di amministratore viene impostato il context admin, in modo che api.ai riconosca l'utente come un potenziale amministratore e gli chieda se vuole entrare nell'area di amministrazione;

Parametri:

* **event:** **LambaEvent**

Parametro contenente, all'interno del campo **body** sotto forma di stringa in formato JSON, un oggetto contenente tutti i dati relativi ad una richiesta di api.ai al **ConversationWebhookService**. Tali dati sono:

```

1 {
2   "id": "String",
3   "lang": "String",
4   "originalRequest": "Object",
5   "result": "ProcessingResult",
6   "sessionId": "String",
7   "status": "StatusObject",
8   "timestamp": "String"
9 }
```

Per la relativa documentazione, consultare la pagina <https://docs.api.ai/docs/query#response>;

* **context:** **LambdaContext**

Parametro utilizzato dal webhook per inviare la risposta. La risposta, contenuta nel **LambdaResponse** parametro del metodo **LambdaContext::succeed**, possiede un attributo **body**, il quale conterrà il corpo di essa sotto forma di una stringa in formato JSON.

La risposta può essere una tra i seguenti tipi:

- l'utente viene riconosciuto come un potenziale amministratore;
- l'utente viene riconosciuto come un ospite conosciuto.

Nel primo caso, la risposta fornita sarà così organizzata:

```

1 {
2   "contexts": [{
3     "name": "String",
4     "first\_name": "String",
5     "last\_name": "String",
6     "username": "String"
7   }]
8 }
```

Dove

- **name** indica il nome del context, che in questo caso sarà "admin";
- **first_name** indica il nome dell'amministratore;
- **last_name** indica il cognome dell'amministratore;
- **username** indica lo username dell'amministratore;

Nel secondo caso, la risposta sarà così organizzata:

```

1 {
2   "contexts": [{
3     "name": "String",
4     "name\_guest": "String",
5     "company": "String",
6     "first\_name": "String",
7     "last\_name": "String"
8   }]
9 }
```

Dove

- **name** indica il nome del context, che in questo caso sarà "welcome";
- **name_guest** indica il nome dell'ospite;
- **company** indica l'azienda di provenienza dell'ospite;
- **username** indica lo username dell'amministratore;
- **first_name** indica il nome della persona desiderata;
- **last_name** indica il cognome della persona desiderata;

;

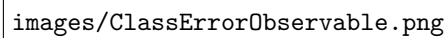
+ <<Create>> createConversationWebhookService(conversation: ConversationsDAO, guest: GuestsDAO, user: UsersDAO): ConversationWebhookService
Metodo che permette la costruzione di un ConversationWebhookService. Permette la dependency injection che ha come oggetti un ConversationsDAO, GuestsDAO e UsersDAO;

Parametri:

- * **conversation:** ConversationsDAO
Attributo contenente il ConversationsDAO;
- * **guest:** GuestsDAO
Attributo contenente il GuestsDAO;
- * **user:** UsersDAO
Attributo contenente lo UsersDAO;

ErrorObservable

- **Nome:** ErrorObservable;
- **Tipo:** Class;
- **Descrizione:** ???;
- **Utilizzo:** ???;
- **Padre:** Observable;



images/ClassErrorObservable.png

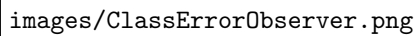
Figura 12: Back-end::ErrorObservable

ErrorObserver

- **Nome:** ErrorObserver;
- **Tipo:** Class;
- **Descrizione:** ???;
- **Utilizzo:** ???;

Guest

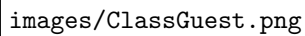
- **Nome:** Guest;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare gli attributi relativi ad un ospite conosciuto, i quali dovranno essere salvati nel database;



images/ClassErrorObserver.png

Figura 13: Back-end::ErrorObserver

- **Utilizzo:** fornisce gli attributi relativi ad un ospite;
- **Attributi:**
 - + **conversations:** **ConversationIdArray**
Attributo contenente l'array delle conversazioni avute con l'ospite;
 - + **name:** **String**
Attributo contenente il nome dell'ospite;
 - + **company:** **String**
Attributo contenente l'azienda di provenienza dell'ospite;
 - + **met:** **StringAssocArray**
Attributo contenente l'array associativo del numero di volte che una persona è stata accolta. La chiave di questo array associativo è il nome dell'ospite;

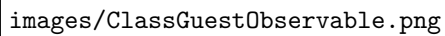


images/ClassGuest.png

Figura 14: Back-end::Guest

GuestObservable

- **Nome:** GuestObservable;
- **Tipo:** Class;
- **Descrizione:** questa classe implementa un **Observable** che permette l'iscrizione di **GuestObserver**;
- **Utilizzo:** fornisce i meccanismi necessari per il passaggio di una serie di **Guest** ad un **Observer** interessato;
- **Padre:** Observable;
- **Metodi:**
 - + <<Create>> createGuestObservable(onSubscription: function(observer: GuestObserver)
: void): GuestObservable
Constructor di GuestObservable;
Parametri:



images/ClassGuestObservable.png

Figura 15: Back-end::GuestObservable

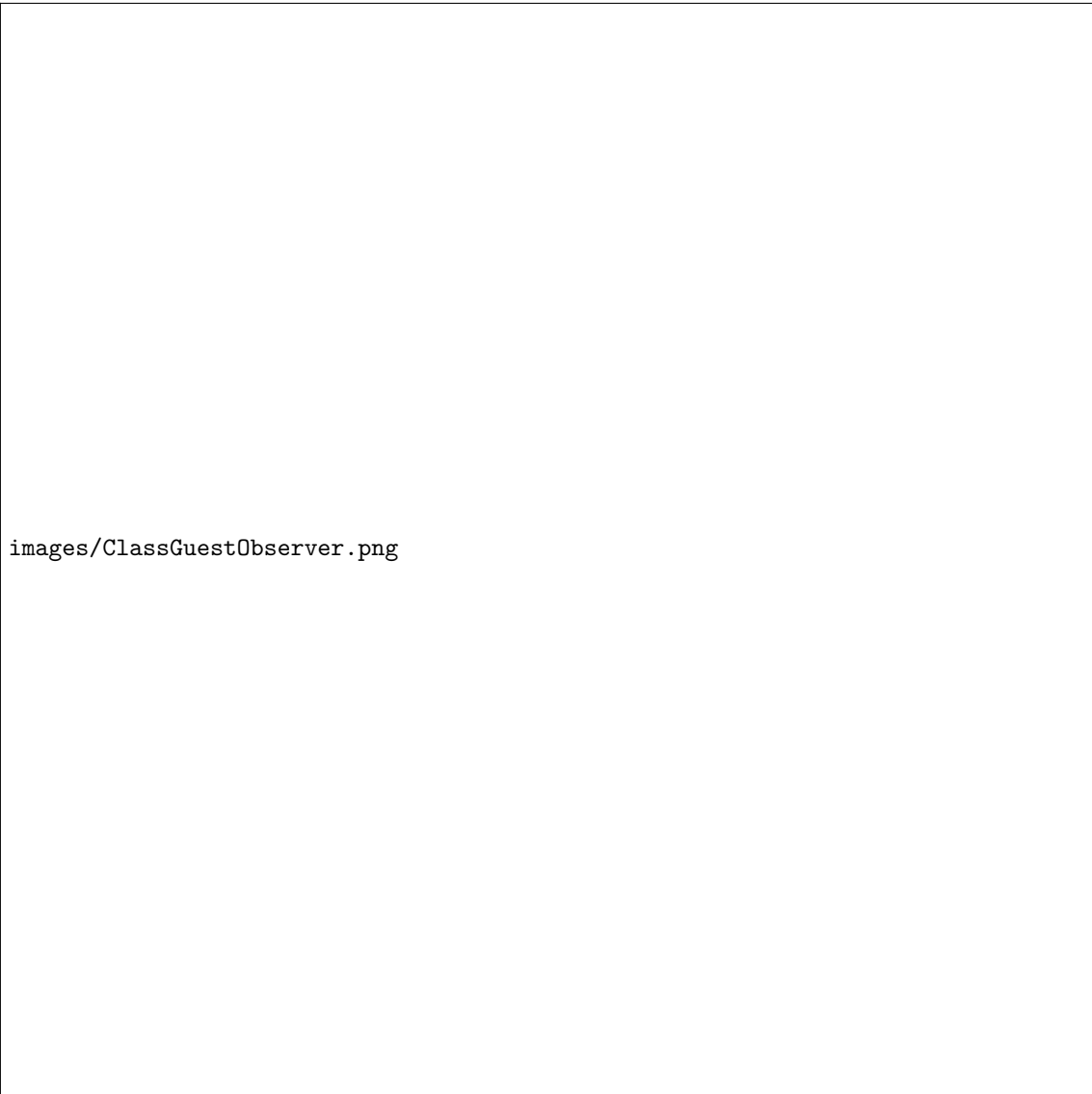
```
* onSubscription: function(observer: GuestObserver) : void
  Funzione che verrà eseguita quando un Observer si iscrive all'Observable. Si oc-
  cupa di passare i dati all'Observer, chiamando il metodo next(guest: Guest).
  Quando non ci sono più dati da restituire, si occupa di chiamare il metodo complete().
  Nel caso in cui si verificasse un errore, si occupa di chiamare il metodo error(err:
  Object) con i dati relativi all'errore verificatosi;

+ subscribe(observer: GuestObserver): Subscription
  Metodo che permette ad un GuestObserver interessato di iscriversi a questo Observable;
  Parametri:

  * observer: GuestObserver
    Observer che si vuole iscrivere;
```

GuestObserver

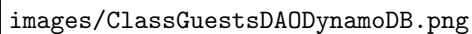
- **Nome:** GuestObserver;



images/ClassGuestObserver.png

Figura 16: Back-end::GuestObserver

- **Tipo:** Class;
- **Descrizione:** classe che rappresenta un **Observer** che si aspetta dati di tipo **Guest**;
- **Utilizzo:** implementa il metodo **next()** dell'interfaccia, in maniera tale che accetti dati di tipo **Guest**;
- **Metodi:**
 - + **next(guest: Guest): void**
Metodo che permette agli **Observable** di notificare l'**Observer** con dati di tipo **Guest**.
Definisce inoltre le operazioni che l'**Observer** compierà all'arrivo di tali dati;
Parametri:
 - * **guest: Guest**
Parametro contenente il **Guest** mandato dall'**Observable**;



images/ClassGuestsDAODynamoDB.png

Figura 17: Back-end::GuestsDAODynamoDB

GuestsDAODynamoDB

- **Nome:** GuestsDAODynamoDB;
- **Tipo:** Class;
- **Descrizione:** classe che si occupa di implementare l'interfaccia **GuestsDAO**, utilizzando un database DynamoDB come supporto per la memorizzazione dei dati;
- **Utilizzo:** implementa i metodi dell'interfaccia **GuestsDAO** interrogando un database DynamoDB. Utilizza **AWS::DynamoDB::DocumentClient** per l'accesso al database. La dependency injection dell'oggetto **AWS::DynamoDB** viene fatta utilizzando il costruttore;
- **Attributi:**
 - **db:** **AWS::DynamoDB**
Attributo contenente un riferimento al modulo di Node.js utilizzato per l'accesso al database DynamoDB contenente la tabella degli utenti;
- **Metodi:**

+ addGuest(guest: Guest): ErrorObservable

Implementazione del metodo definito nell'interfaccia `GuestsDAO`. Utilizza il metodo `put` del `DocumentClient` per aggiungere l'ospite al database;

Parametri:

* guest: Guest

Parametro contenente l'ospite da aggiungere;

+ getGuest(name: String, company: String): GuestObservable

Implementazione del metodo definito nell'interfaccia `GuestsDAO`. Utilizza il metodo `get` del `DocumentClient` per ottenere i dati relativi ad un `Guest` dal database;

Parametri:

* name: String

Parametro contenente il nome dell'ospite;

* company: String

Parametro contenente l'azienda di provenienza dell'ospite;

+ getGuestList(): GuestObservable

Implementazione del metodo dell'interfaccia `GuestsDAO`. Utilizza il metodo `scan` del `DocumentClient` per ottenere la lista degli ospiti dal database;

+ removeGuest(name: String, company: String): ErrorObservable

Implementazione del metodo dell'interfaccia `GuestsDAO`. Utilizza il metodo `delete` del `DocumentClient` per eliminare un ospite dal database;

Parametri:

* name: String

Parametro contenente il nome dell'ospite;

* company: String

Parametro contenente l'azienda di provenienza dell'ospite;

+ updateGuest(guest: Guest): ErrorObservable

Implementazione del metodo dell'interfaccia `GuestsDAO`. Utilizza il metodo `update` del `DocumentClient` per aggiornare i dati relativi ad un ospite presenti all'interno del database;

Parametri:

* guest: Guest

Parametro contenente l'ospite da aggiornare;

+ <> createGuestsDAODynamoDB(db: AWS::DynamoDB): GuestsDAODynamoDB

Constructor della classe `GuestsDAODynamoDB`. Permette di effettuare la dependency injection di `AWS::DynamoDB`;

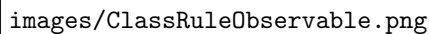
Parametri:

* db: AWS::DynamoDB

Parametro contenente un riferimento al modulo di Node.js da utilizzare per l'accesso al database `DynamoDB` contenente la tabella degli ospiti;

RuleObservable

- **Nome:** `RuleObservable`;
- **Tipo:** `Class`;
- **Descrizione:** questa classe implementa un `Observable` che permette l'iscrizione di `RuleObserver`;
- **Utilizzo:** fornisce i meccanismi necessari per il passaggio di una serie di `Rule` ad un `Observer` interessato;



images/ClassRuleObservable.png

Figura 18: Back-end::RuleObservable

- **Padre:** Observable;

- **Metodi:**

```
+ <<Create>> createRuleObservable(onSubscription: function(observer: RuleObserver)
: void): RuleObservable
```

Constructor di RuleObservable;

Parametri:

```
* onSubscription: function(observer: RuleObserver) : void
```

Funzione che verrà eseguita quando un **Observer** si iscrive all'**Observable**. Si occupa di passare i dati all'**Observer**, chiamando il metodo **next(rule: Rule)**. Quando non ci sono più dati da restituire, si occupa di chiamare il metodo **complete()**. Nel caso in cui si verificasse un errore, si occupa di chiamare il metodo **error(err: Object)** con i dati relativi all'errore verificatosi;

```
+ subscribe(observer: RuleObserver): Subscription
```

Metodo che permette ad un **RuleObserver** interessato di iscriversi a questo **Observable**;

Parametri:

```
* observer: RuleObserver
Observer che si vuole iscrivere;
```

RuleObserver

images/ClassRuleObserver.png

Figura 19: Back-end::RuleObserver

- **Nome:** RuleObserver;
- **Tipo:** Class;
- **Descrizione:** classe che rappresenta un **Observer** che si aspetta dati di tipo **Rule**;
- **Utilizzo:** implementa il metodo **next()** dell'interfaccia, in maniera tale che accetti dati di tipo **Rule**;
- **Metodi:**
 - + **next(rule: Rule): void**
Metodo che permette agli **Observable** di notificare l'**Observer** con dati di tipo **Rule**.

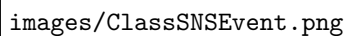
Definisce inoltre le operazioni che l'**Observer** compierà all'arrivo di tali dati;

Parametri:

* **rule:** **Rule**

Parametro contenente la **Rule** mandata dall'**Observable**;

SNSEvent



images/ClassSNSEvent.png

Figura 20: Back-end::SNSEvent

- **Nome:** SNSEvent;
- **Tipo:** Class;
- **Descrizione:** questa classe rappresenta l'oggetto ricevuto da una lambda function in seguito alla pubblicazione di un messaggio su un topic di SNS a cui tale funzione è iscritta;
- **Utilizzo:** fornisce gli attributi relativi ad una notifica mandata da SNS ad una lambda function iscritta ad un topic sul quale sia stato pubblicato un messaggio;
- **Attributi:**

+ **Records:** SNSRecordArray

Array contenente i dati della notifica mandata da SNS alla lambda function. Contiene un unico oggetto di tipo Record;

SNSMessage

images/ClassSNSMessage.png

Figura 21: Back-end::SNSMessage

- **Nome:** SNSMessage;
- **Tipo:** Class;
- **Descrizione:** questa classe rappresenta un messaggio mandato da SNS in seguito ad un'interazione con l'assistente virtuale;
- **Utilizzo:** fornisce un meccanismo event-driven per la gestione dei dati relativi alle interazioni col sistema.
Per la relativa documentazione, consultare la pagina <http://docs.aws.amazon.com/sns/latest/dg/json-formats.html#http-subscription-confirmation-json>;
- **Attributi:**

```
+ MessageId: String
Attributo contenente l'identificativo del messaggio;

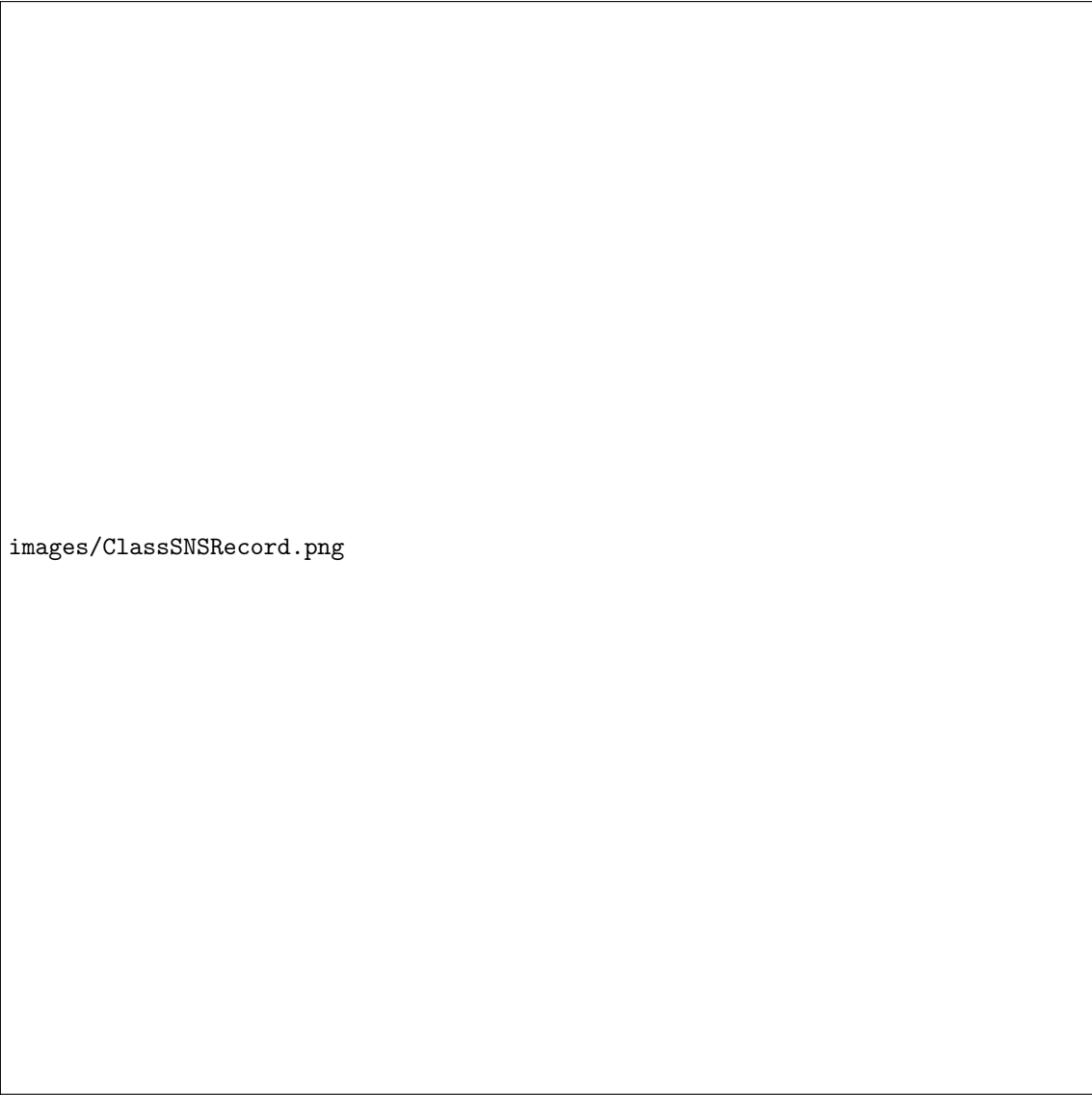
+ Subject: String
Attributo contenente il Subject che dev'essere notificato dal sistema SNS;

+ Message: String
Attributo contenente il messaggio da pubblicare;

+ Timestamp: String
Attributo contenente il timestamp relativo al momento della pubblicazione del messaggio;

+ MessageAttributes: SnsMessageAttributes
Attributo contenente;
```

SNSRecord



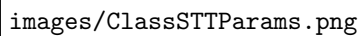
images/ClassSNSRecord.png

Figura 22: Back-end::SNSRecord

- **Nome:** SNSRecord;

- **Tipo:** Class;
- **Descrizione:** questa classe rappresenta uno dei records mandati da SNS ad una lambda function. ;
- **Utilizzo:** fornisce gli attributi di un record mandato da SNS ad una lambda function. ;
- **Attributi:**
 - + Sns: SNSMessage
 - ;

STTPParams



images/ClassSTTPParams.png

Figura 23: Back-end::STTPParams

- **Nome:** STTPParams;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare ed organizzare i parametri necessari a richiamare le API Watson Speech to Text;

- **Utilizzo:** fornisce l'insieme dei parametri necessari ad identificare un audio, nel formato richiesto dall'API Watson Speech to Text, che verrà poi convertito in testo. Per la relativa documentazione, consultare la pagina https://www.ibm.com/watson/developercloud/speech-to-text/api/v1/#recognize_audio_websockets;

- **Attributi:**

- + **audio:** `ReadStream`

- Attributo contenente lo stream per la lettura del file audio;

- + **content_type:** `String`

- Attributo contenente il formato dell'audio;

- + **max_alternatives:** `Integer[0..1]`

- Attributo contenente il numero massimo di alternative testuali che devono essere fornite per il relativo audio. Il valore di default di questo attributo è uguale a 1;

- + **timestamps:** `Boolean[0..1]`

- Attributo contenente un valore booleano che indica se ottenere un timestamp per ogni parola. Il valore di default per questo attributo è false;

- + **word_confidence:** `Boolean[0..1]`

- Attributo contenente un valore booleano che indica se ottenere il grado di confidenza, contenuto nell'intervallo [0,1], per ogni parola. Il valore di default per questo attributo è false;

- + **inactivity_timeout:** `Integer[0..1]`

- Attributo contenente il tempo in secondi dopo il quale, se nell'audio viene rilevato solo del silenzio, la connessione deve essere chiusa. Il valore di default di questo attributo è 30 secondi.

- Per non chiudere mai la connessione, si può impostare questo attributo a -1. ;

- + **model:** `String[0..1]`

- Attributo contenente il nome del model relativo all'audio da trascrivere.

- Un model è un parametro che indica la lingua e il tipo di sampling rates usato per essa.

- Il tipo del sampling rates supportato può assumere uno tra i seguenti valori:

- * `broadband`;

- * `narrowband`.

- Si rimanda alla relativa documentazione (https://www.ibm.com/watson/developercloud/speech-to-text/api/v1/?curl#get_models) per ulteriori chiarimenti;

- + **interim_results:** `Boolean[0..1]`

- Attributo contenente un valore booleano che indica se posso essere ritornati risultati parziali o meno. Se impostato a true, i risultati parziali sono ritornati come uno stream di oggetti JSON ed ognuno di essi rappresenta un singolo `SpeechRecognitionEvent`. Se impostato a false, verrà ritornato un unico `SpeechRecognitionEvent` contenente il risultato finale.

- Questo attributo ha valore di default uguale a false;

- + **keywords:** `StringArray[0..1]`

- Attributo contenente l'array delle parole da ricercare nell'audio. Ogni cella di questo array può contenere una o più parole da cercare. ;

- + **keywords_threshold:** `Real[0..1]`

- Attributo contenente un valore di confidenza, contenuto nell'intervallo [0,1], che è il limite inferiore per una keyword trovata. Una parola fa match in una keyword se la sua confidenza è maggiore o uguale al valore di questo attributo.

- Se questo parametro è omesso, nessuna keyword sarà trovata, altrimenti deve essere fornita almeno una keyword;

+ **word_alternatives_threshold:** Real[0..1]

Attributo contenente un valore di confidenza, contenuto nell'intervallo [0,1], che è il limite inferiore per identificare un'ipotetica parola come possibile alternativa.

Una parola alternativa è considerata tale se la sua confidenza è maggiore o uguale al valore di questo attributo. Se questo parametro è omissso, nessuna parola alternativa verrà fornita;

+ **profanity_filter:** Boolean[0..1]

Attributo contenente un valore booleano che indica se il profanity filter verrà applicato al testo trascritto.

Il profanity filter è un meccanismo che sostituisce parole inappropriate con degli asterischi. Questo filtro può essere applicato solo a trascrizioni in lingua US English.

Il valore di default per questo attributo è uguale a true;

+ **smart_formatting:** Boolean[0..1]

Attributo contenente un valore booleano che indica se date, orari, serie di numeri e cifre, numeri di telefono, valori monetari e indirizzi Internet devono essere convertiti, nella trascrizione finale, in un formato più leggibile. Questo meccanismo può essere applicato solo a trascrizioni in lingua US English.

Il valore di default per questo attributo è uguale a false;

+ **customization_id:** String[0..1]

Attributo contenente il GUID relativo al model personalizzato utilizzato. Per default, nessun modello personalizzato è utilizzato;

+ **speaker_labels:** Boolean[0..1]

Indicates whether labels that identify which words were spoken by which participants in a multi-person exchange are to be included in the response. If true, speaker labels are returned; if false (the default), they are not. Speaker labels can be returned only for the following language models: en-US_NarrowbandModel es-ES_NarrowbandModel ja-JP_NarrowbandModel Setting speaker_labels to true forces the continuous and time-stamps parameters to be true, as well, regardless of whether the user specifies false for the parameters. For more information, see Speaker labels;

TaskObservable

- **Nome:** TaskObservable;
- **Tipo:** Class;
- **Descrizione:** questa classe implementa un **Observable** che permette l'iscrizione di **TaskObserver**;
- **Utilizzo:** fornisce i meccanismi necessari per il passaggio di una serie di **Task** ad un **Observer** interessato;
- **Padre:** Observable;
- **Metodi:**

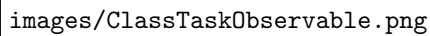
+ <<Create>> createTaskObservable(onSubscription: function(observer: TaskObserver) : void): TaskObservable

Constructor di TaskObservable;

Parametri:

* onSubscription: function(observer: TaskObserver) : void

Funzione che verrà eseguita quando un **Observer** si iscrive all'**Observable**. Si occupa di passare i dati all'**Observer**, chiamando il metodo **next(function: Task)**. Quando non ci sono più dati da restituire, si occupa di chiamare il metodo **complete()**. Nel caso in cui si verificasse un errore, si occupa di chiamare il metodo **error(err: Object)** con i dati relativi all'errore verificatosi;



images/ClassTaskObservable.png

Figura 24: Back-end::TaskObservable

```
+ subscribe(observer: TaskObserver): void
```

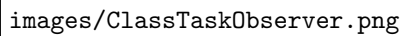
Metodo che permette ad un **TaskObserver** interessato di iscriversi a questo **Observable**;

Parametri:

```
* observer: TaskObserver  
  Observer che si vuole iscrivere;
```

TaskObserver

- **Nome:** TaskObserver;
- **Tipo:** Class;
- **Descrizione:** classe che rappresenta un **Observer** che si aspetta dati di tipo **Task**;
- **Utilizzo:** implementa il metodo **next()** dell'interfaccia, in maniera tale che accetti dati di tipo **Task**;
- **Metodi:**



images/ClassTaskObserver.png

Figura 25: Back-end::TaskObserver

```
+ next(task: Task): void
```

Metodo che permette agli **Observable** di notificare l'**Observer** con dati di tipo **Task**.

Definisce inoltre le operazioni che l'**Observer** compierà all'arrivo di tali dati;

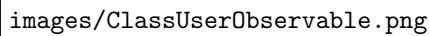
Parametri:

```
* task: Task
```

Parametro contenente la **Task** mandata dall'**Observable**;

UserObservable

- **Nome:** UserObservable;
- **Tipo:** Class;
- **Descrizione:** questa classe implementa un **Observable** che permette l'iscrizione di **UserObserver**;
- **Utilizzo:** fornisce i meccanismi necessari per il passaggio di una serie di **User** ad un **Observer** interessato;



images/ClassUserObservable.png

Figura 26: Back-end::UserObservable

- **Padre:** Observable;

- **Metodi:**

- + `subscribe(observer: UserObserver): Subscription`

Metodo che permette ad uno `UserObserver` interessato di iscriversi a questo `Observable`;

Parametri:

- * `observer: UserObserver`
Observer che si vuole iscrivere;

- + `<<Create>> createUserObservable(onSubscription: function(observer: UserObserver) : void): UserObservable`

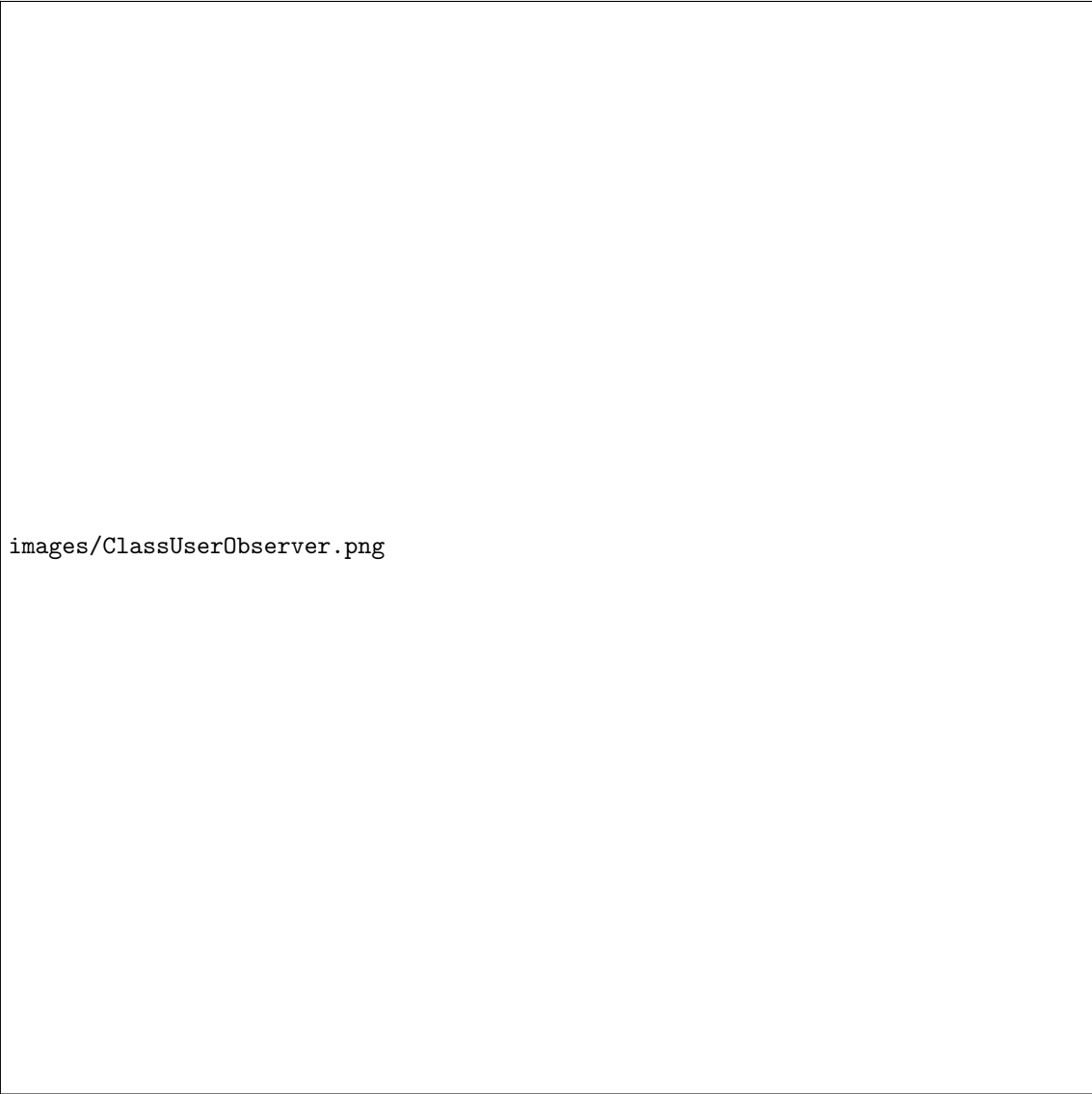
Constructor di `UserObservable`;

Parametri:

- * `onSubscription: function(observer: UserObserver) : void`
Funzione che verrà eseguita quando un `Observer` si iscrive all'`Observable`. Si occupa di passare i dati all'`Observer`, chiamando il metodo `next(user: User)`. Quan-

do non ci sono più dati da restituire, si occupa di chiamare il metodo `complete()`. Nel caso in cui si verificasse un errore, si occupa di chiamare il metodo `error(err: Object)` con i dati relativi all'errore verificatosi;

UserObserver



images/ClassUserObserver.png

Figura 27: Back-end::UserObserver

- **Nome:** `UserObserver`;
- **Tipo:** `Class`;
- **Descrizione:** classe che rappresenta un `Observer` che si aspetta dati di tipo `User`. ;
- **Utilizzo:** implementa il metodo `next()` dell'interfaccia, in maniera tale che accetti dati di tipo `User`;
- **Metodi:**
 - + `next(user: User): void`
Metodo che permette agli `Observable` di notificare l'`Observer` con dati di tipo `User`.

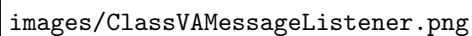
Definisce inoltre le operazioni che l'**Observer** compierà all'arrivo di tali dati;

Parametri:

* **user:** **User**

Parametro contenente lo **User** mandato dall'**Observable**;

VAMessageListener



images/ClassVAMessageListener.png

Figura 28: Back-end::VAMessageListener

- **Nome:** VAMessageListener;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di registrare i dati relativi alle interazioni degli ospiti col nostro sistema;
- **Utilizzo:** fornisce un meccanismo per registrare i dialoghi che gli ospiti hanno con il sistema. Fornisce una lambda function che quando viene generato un evento **SNSMessage** in seguito all'arrivo di una risposta da parte dell'assistente virtuale, si occupa di registrare i dati della

relativa interazione tramite **ConversationsDAO**, ed eventualmente di aggiornare i dati relativi all'ospite utilizzando **GuestsDAO**.

SNS chiama tale lambda function con un oggetto del tipo **SNSEvent**, il quale contiene al suo interno un array di **SNSRecord**. Questo array in realtà ha un unico oggetto, il quale al suo interno contiene l'**SNSMessage** inviato.

Di seguito viene riportato un esempio dell'oggetto utilizzato.

```

1 {
2   "Records":
3   [
4     {
5       "Sns":
6       {
7         "Message": "Corpo del messaggio pubblicato sul topic di
8           SNS",
9         "MessageAttributes":{"key": "value", "key2": "value2"},
10        "MessageId":"stringa-contenente-id-del-messaggio",
11        "Subject":"oggetto del messaggio",
12        "Timestamp":"2017-03-26T20:39:48.599Z"
13      }
14    ]
15  }

```

;

- **Attributi:**

- **guests:** **GuestsDAO**
Attributo che permette di contattare il **GuestsDAO**;
- **conversations:** **ConversationDAO**
Attributo che permette di contattare il **ConversationsDAO**;

- **Eventi gestiti:**

- **SNSEvent**
Messaggio mandato ad sns quando arriva la risposta dall'assistente virtuale. All'arrivo del messaggio si occupa di salvare i dati dell'interazione nel DAO.

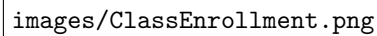
Back-end::APIGateway

Package contenente le componenti necessarie a gestire le richieste ricevute dal backend.

Classi

Enrollment

- **Nome:** **Enrollment**;
- **Tipo:** **Class**;
- **Descrizione:** questa classe fornisce gli attributi necessari al passaggio di un Enrollment alle lambda function;
- **Utilizzo:** fornisce gli attributi relativi ad un Enrollment;
- **Attributi:**



images/ClassEnrollment.png

Figura 29: Back-end::APIGateway::Enrollment

+ username: String

Attributo contenente l'username dell'utente associato all'Enrollment;

+ audio: Base64String

Attributo contenente la traccia audio che sarà oggetto dell'Enrollment, codificata in Base64;

VARestAPIBody DA MODIFICARE DA MODIFICARE DA MODIFICARE DA MODIFICARE

- **Nome:** VARestAPIBody DA MODIFICARE DA MODIFICARE DA MODIFICARE DA MODIFICARE;
- **Tipo:** Class;
- **Descrizione:** questa classe fornisce gli attributi necessari al client per effettuare una richiesta all'API REST del back-end;

- **Utilizzo:** fornisce gli attributi relativi ad una richiesta all'API REST del back-end e viene utilizzata dalla classe `VARestAPIEvent`. DA MODIFICARE DA MODIFICARE DA MODIFICARE DA MODIFICARE DA MODIFICARE (manca la classe `VARestAPIEvent`);
- **Attributi:**
 - + `app: String`
Attributo contenente il nome dell'applicazione da cui arriva la richiesta;
 - + `session_id: String`
Attributo contenente l'id della sessione corrente, creato dal Client;
 - + `audio: Base64String`
Attributo contenente l'audio della richiesta codificata in Base64;
 - + `data: ObjectAssocArray`
Attributo contenente un array associativo di `Object` ricevuti dall'Assistente Virtuale;

VocalAPI

- **Nome:** `VocalAPI`;
- **Tipo:** `Class`;
- **Descrizione:** questa classe si occupa di implementare l'endpoint dell'API Gateway utilizzato dal client vocale;
- **Utilizzo:** grazie al metodo pubblico di questa classe, fornisce un meccanismo che:
 - permette di dedurre, tramite i microservizi di STT e di assistente virtuale, il servizio necessario al client vocale, utilizzando il metodo `queryLambda`;
 - permette al client vocale di usufruire delle funzionalità, supportate dal `Backend`, tramite i metodi privati che questa classe fornisce.
- ;
- **Attributi:**
 - `vocal: VocalLoginModule`
Attributo contenente il `VocalLoginModule` di cui è stata eseguita la dependency injection nel costruttore. Viene utilizzato per effettuare il login nel servizio di Speaker Recognition;
 - `stt: STTModule`
Attributo contenente il modulo utilizzato per contattare le API per il servizio di Watson Speech to Text di IBM;
 - `sns: AWS::SNS`
Attributo che permette di contattare il servizio SNS;
 - `jwt: JSONWebTokenModule`
Attributo contenente il `JSONWebTokenModule` di cui è stata eseguita la dependency injection nel costruttore. Viene utilizzato per creare un `JSONWebToken` in caso di autenticazione al sistema avvenuta con successo;
 - `request_promise: RequestPromiseModule`
Attributo contenente il `RequestPromiseModule` di cui è stata eseguita la dependency injection nel costruttore. Viene utilizzato per effettuare richieste HTTP sostituendo callbacks con promises;
- **Metodi:**

```
+ <<Create>> createAuthAPI(vocal: VocalLoginModule, jwt: JSONWebTokenModule,
rp: RequestPromiseModule): AuthAPI
```

Costruttore della classe `AuthAPI` che permette la dependency injection di `VocalLoginModule`;
Parametri:

```
* vocal: VocalLoginModule
```

Parametro che permette di effettuare la dependency injection di `VocalLoginModule`;

```
* jwt: JSONWebTokenModule
```

Parametro che permette di effettuare la dependency injection di `JSONWebTokenModule`;

```
* rp: RequestPromiseModule
```

Parametro che permette di effettuare la dependency injection di `RequestPromiseModule`;

```
- addUserEnrollment(enr: Enrollment): Error
```

Metodo che permette di aggiungere un enrollment ad un utente del sistema. Restituisce un oggetto di tipo `Error`, con code impostato a 1 nel caso in cui l'utente non esista;

Parametri:

```
* enr: Enrollment
```

Parametro contenente l'enrollment da aggiungere a un utente;

```
- addUser(user: User): Error
```

Metodo che permette di aggiungere un utente al sistema. Restituisce un oggetto di tipo `Error`, con code impostato a 1 in caso di username già esistente, 2 in caso di username non valido (troppo lungo o troppo corto);

Parametri:

```
* user: User
```

Parametro contenente l'user che si vuole aggiungere al sistema;

```
- getUser(username: String): Error
```

Metodo che permette di ottenere i dati relativi ad un utente del sistema. Restituisce l'oggetto `User` relativo all'utente con lo username indicato. In caso tale utente non esista, restituisce un oggetto vuoto;

Parametri:

```
* username: String
```

Parametro contenente l'username dell'utente del quale si vogliono ottenere i dati;

```
- getUserList(): UserArray
```

Metodo che permette di ottenere una lista degli utenti del sistema;

```
- loginUser(enr: Enrollment): String
```

Metodo che si occupa di gestire il login vocale degli utenti. Restituisce una stringa contenente il JWT in caso di autenticazione avvenuta con successo, altrimenti restituisce una stringa vuota;

Parametri:

```
* enr: Enrollment
```

Attributo contenente l'`Enrollment` (audio + username) con il quale tentare il login;

```
- removeUser(username: String): Error
```

Metodo che permette di eliminare i dati relativi ad un utente dal sistema. Restituisce un oggetto di tipo `Error`, con code impostato a 1 nel caso in cui l'utente non esista;

Parametri:

```
* username: String
```

Parametro contenente l'username dell'user da eliminare dal sistema;

```
- resetUserEnrollment(username: String): Error
```

Metodo che permette di eliminare tutti gli enrollments di un utente del sistema. Re-

stituisce un oggetto di tipo `Error`, con code impostato a 1 nel caso in cui l'utente non esista;

Parametri:

* `username: String`

Parametro contenente l'username dell'utente a cui si vogliono eliminare tutti gli enrollments;

- `updateUser(user: User): Error`

Metodo che permette di modificare i dati relativi ad un utente del sistema. Restituisce un oggetto di tipo `Error`, con code impostato a 1 nel caso in cui l'utente non esista;

Parametri:

* `user: User`

Parametro contenente l'user da modificare;

- `addRule(rule: Rule): Error`

Metodo che permette di aggiungere una direttiva al sistema. Restituisce un oggetto di tipo `Error`;

Parametri:

* `rule: Rule`

Parametro contenente la `Rule`;

- `getRule(id: String): Rule`

Metodo che permette di ottenere i dati relativi ad una direttiva del sistema a partire dal suo id. Restituisce la direttiva in questione, oppure un oggetto vuoto nel caso in cui tale direttiva non esista;

Parametri:

* `id: String`

Parametro contenente l'identificativo della `Rule`;

- `getRuleList(): RuleArray`

Metodo che permette di ottenere la lista delle direttive del sistema. ;

- `updateRule(rule: Rule): Error`

Metodo che permette di aggiornare una direttiva presente nel sistema. Restituisce un oggetto di tipo `Error`;

Parametri:

* `rule: Rule`

Parametro contenente la `Rule` aggiornata;

- `removeRule(id: String): Error`

Metodo che permette di rimuovere una direttiva presente nel sistema. Restituisce un oggetto di tipo `Error`;

Parametri:

* `id: String`

Parametro contenente l'identificativo della `Rule`;

+ `queryLambda(event: LambdaEvent, context: LambdaContext): void`

Metodo che si occupa di chiamare prima il servizio di Speech To Text e, una volta ottenuta risposta da esso, di interrogare l'assistente virtuale. Quando viene ricevuta una risposta dall'assistente virtuale, il metodo controlla il valore del campo `action` di tale risposta e, nel caso in cui corrisponda ad una delle action supportate, si occupa di eseguire le azioni necessarie utilizzando i metodi privati di questa classe. Nel caso in cui invece `action` non corrisponda ad una delle azioni supportate (ovvero `action` è un'azione che non richiede operazione da parte del back-end, oppure `actionIncomplete` è impostato a true), tale risposta viene rielaborata ed inoltrata. Le azioni supportate

ed i relativi compiti da svolgere sono disponibili alla sezione [hyperef a sezione].
 Ad ogni interazione viene inoltre pubblicato un messaggio su un topic di sns, utilizzando il metodo `sns.publish()`, in modo che sia possibile registrare i dati relativi a tali interazioni;

Parametri:

* `event: LambdaEvent`

Parametro contenente, all'interno del campo `body` sotto forma di stringa in formato JSON, un oggetto contenente tutti i dati relativi ad un messaggio da inviare. Tali dati sono:

```
1 {
2
3 }
```

;

* `context: LambdaContext`

Parametro utilizzato dalle lambda function per inviare la risposta. La risposta, contenuta nel `LambdaResponse` parametro del metodo `LambdaContext::succeed`, possiede un attributo `body`, il quale conterrà il corpo di essa sotto forma di una stringa in formato JSON, organizzando i dati nel seguente modo:

```
1 {
2   "transcript": "String",
3   "confidence": "Number",
4   "timestamps": "StringArray",
5   "word\_confidence": "StringArray"
6 }
```

Per la relativa documentazione, consultare la pagina https://www.ibm.com/watson/developercloud/speech-to-text/api/v1/#recognize_sessionless_nonmp12. ;

VocalLoginModuleConfig

- **Nome:** `VocalLoginModuleConfig`;
- **Tipo:** `Class`;
- **Descrizione:** questa classe viene utilizzata per la configurazione di `VocalLoginModule`;
- **Utilizzo:** fornisce gli attributi necessari alla configurazione di `VocalLoginModule`;
- **Attributi:**

+ `min_confidence: int`

Questo attributo indica la confidence minima richiesta perchè il login avvenga con successo;

Back-end::Auth

Package contenente le componenti del microservizio necessario all'autenticazione.

Classi

UsersDAO

- **Nome:** `UsersDAO`;

- **Tipo:** Interface;
- **Descrizione:** questa classe si occupa di astrarre le modalità d'interazione al database per questo microservizio. ;
- **Utilizzo:** fornisce a `UserService` un meccanismo per accedere al database contenente gli utenti registrati, senza conoscerne le modalità di implementazione e di persistenza di quest'ultimo. A partire da un identificativo, permette operazioni di lettura, scrittura e rimozione di utenti registrati;
- **Figlio:** `UsersDAODynamoDB`;
- **Metodi:**

+ `addUser(user: User): ErrorObservable`

Metodo che permette di aggiungere un utente.

L'`Observable` restituito non riceverà alcun valore, ma verrà completato in caso di aggiunta dell'utente avvenuta con successo. In caso di errore durante l'aggiunta dell'utente, gli `Observer` interessati verranno notificati tramite la chiamata del loro metodo `error()` con i dati relativi all'errore verificatosi;

Parametri:

* `user: User`

Parametro contenente un utente registrato;

+ `removeUser(id: String): ErrorObservable`

Metodo che permette di rimuovere un utente registrato. L'`Observable` restituito non riceverà alcun valore, ma verrà completato in caso di aggiunta dell'utente avvenuta con successo. In caso di errore durante l'aggiunta dell'utente, gli `Observer` interessati verranno notificati tramite la chiamata del loro metodo `error()` con i dati relativi all'errore verificatosi;

Parametri:

* `id: String`

Parametro contenente l'id dello `User` che si vuole eliminare;

+ `getUser(username: String): UserObservable`

Metodo che permette di ottenere i dati relativi ad un utente.

L'`Observable` restituito riceverà l'oggetto rappresentante tale `User`, e verrà completato. Nel caso in cui l'utente richiesto non sia presente nel database, gli `Observer` interessati non riceveranno alcun valore, ma verranno notificati tramite la chiamata del loro metodo `error()`;

Parametri:

* `username: String`

Parametro contenente lo username dello `User` che si vuole ottenere;

+ `getUserList(): UserObservable`

L'`Observable` restituito manderà agli `Observer` gli utenti ottenuti, uno alla volta, e poi chiama il loro metodo `complete`. Nel caso in cui si verifichi un errore, gli `Observer` iscritti verranno notificati tramite la chiamate del loro metodo `error` con i dati relativi all'errore verificatosi;

+ `updateUser(user: User): ErrorObservable`

Metodo che permette di aggiornare un utente registrato. L'`Observable` restituito non riceverà alcun valore, ma verrà completato in caso di aggiunta dell'utente avvenuta con successo. In caso di errore durante l'aggiunta dell'utente, gli `Observer` interessati verranno notificati tramite la chiamata del loro metodo `error()` con i dati relativi all'errore verificatosi;

Parametri:

* **user:** **User**
Parametro contenente lo **User** aggiornato;

SRUser

- **Nome:** SRUser;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare ed organizzare i parametri dell'autenticazione tramite Speaker Recognition (SR);
- **Utilizzo:** fornisce l'insieme dei parametri necessari ad identificare, tramite SR, un utente sottoposto alla procedura di Enrollment del servizio di Speaker Recognition. Per la relativa documentazione consultare la pagina <https://www.microsoft.com/cognitive-services/en-us/speaker-recognition-api/documentation>. ;
- **Attributi:**
 - + **id:** **String**
Attributo contenente l'identificativo del profilo utente del microservizio di Speaker Recognition;
 - + **enrollmentCount:** **int**
Attributo contenente il numero di Enrollment dello **User**;
 - + **createdDateTime:** **String**
Attributo contenente la data di creazione del profilo utente nel microservizio esterno di Speaker Recognition;
 - + **enrollmentStatus:** **String**
Attributo contenente lo stato dell'Enrollment.
Lo stato può essere uno tra i seguenti valori:
 - * **Enrolling:** indica che la fase di enrollment è in corso;
 - * **Training:** indica che il microservizio di Speaker Recognition sta elaborando e organizzando i dati ricevuti (analizza le frasi comunicate e ne costruisce un'impronta vocale);
 - * **Enrolled,** ovvero che le due fasi precedenti sono state completate.

User

- **Nome:** User;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare i dati relativi ad un utente registrato;
- **Utilizzo:** fornisce i metodi getter e setter per i parametri relativi ad un utente registrato, i quali dovranno essere memorizzati nel database per questo microservizio.
È utilizzata dalla classe **UsersDAO** e dalle classi che utilizzano quest'ultima;
- **Attributi:**
 - **username:** **String**
Attributo contenente l'username dell'utente registrato;
 - **first_name:** **String**
Attributo contenente il nome dell'utente registrato;

- `last_name: String`
Attributo contenente il cognome dell'utente registrato;
- `password: String[0..1]`
Attributo contenente la password dell'utente registrato;
- `slack_channel: String[0..1]`
Attributo contenente il canale Slack dell'utente registrato;
- `sr_id: String[0..1]`
Attributo contenente l'id del profilo utente nel microservizio esterno di Speaker Recognition;

- **Metodi:**

- + `getId(): String`
Metodo che permette di ottenere l'id dell'utente registrato;
- + `setId(id: String): void`
Metodo che permette di impostare l'id dell'utente registrato;
Parametri:
 - * `id: String`
Parametro contenente l'id;
- + `getUsername(): String`
Metodo che permette di ottenere lo username dell'utente registrato;
- + `setUsername(username: String): void`
Metodo che permette di impostare lo username dell'utente registrato;
Parametri:
 - * `username: String`
Parametro relativo all'username da settare;
- + `getNome(): String`
Metodo che permette di ottenere il nome dell'utente registrato;
- + `setNome(nome: String): void`
Metodo che permette di impostare il nome dell'utente registrato;
Parametri:
 - * `nome: String`
Parametro contenente il nome;
- + `getPassword(): String[0..1]`
Metodo che permette di ottenere la password dell'utente registrato;
- + `setPassword(password: String[0..1]): void`
Metodo che permette di impostare la password dell'utente registrato;
Parametri:
 - * `password: String[0..1]`
Parametro relativo alla password da settare;
- + `getSlackChannel(): String[0..1]`
Metodo che permette di ottenere il canale Slack dell'utente registrato;
- + `setSlackChannel(slack_channel: String[0..1]): void`
Metodo che permette di impostare il canale Slack dell'utente registrato necessario per

contattarlo;

Parametri:

* `slack_channel: String[0..1]`

Parametro relativo al canale Slack da settare;

+ `getSrId(): String[0..1]`

Metodo che permette di ottenere l'id del profilo utente nel microservizio esterno di Speaker Recognition;

+ `setSrId(sr_id: String[0..1]): void`

Metodo che permette di impostare l'id dello Speaker Recognition associato all'utente registrato;

Parametri:

* `sr_id: String[0..1]`

Parametro relativo all'id dello Speaker Recognition da settare;

UsersDAODynamoDB

- **Nome:** UsersDAODynamoDB;
- **Tipo:** Class;
- **Descrizione:** classe che si occupa di implementare l'interfaccia `UsersDAO`, utilizzando un database DynamoDB come supporto per la memorizzazione dei dati;
- **Utilizzo:** implementa i metodi dell'interfaccia `UsersDAO` interrogando un database DynamoDB. Utilizza `AWS::DynamoDB::DocumentClient` per l'accesso al database. La dependency injection dell'oggetto `AWS::DynamoDB` viene fatta utilizzando il costruttore;
- **Padre:** <<interface>> `UsersDAO`;
- **Attributi:**
 - `db: AWS::DynamoDB`
Attributo contenente un riferimento al modulo di Node.js utilizzato per l'accesso al database DynamoDB contenente la tabella degli utenti;
- **Metodi:**
 - + `addUser(user: User): ErrorObservable`
Implementazione del metodo definito nell'interfaccia `UsersDAO`. Utilizza il metodo `put` del `DocumentClient` per aggiungere l'utente al database;
Parametri:
 - * `user: User`
Utente che si vuole aggiungere al sistema;
 - + `getUser(username: String): UserObservable`
Implementazione del metodo definito nell'interfaccia `UsersDAO`. Utilizza il metodo `get` del `DocumentClient` per ottenere i dati relativi ad uno `User` dal database;
Parametri:
 - * `username: String`
Parametro contenente lo username dello `User` che si vuole ottenere;
 - + `getUserList(): UserObservable`
Implementazione del metodo dell'interfaccia `UsersDAO`. Utilizza il metodo `scan` del `DocumentClient` per ottenere la lista degli utenti dal database;

```
+ removeUser(username: String): ErrorObservable
```

Implementazione del metodo dell'interfaccia `UsersDAO`. Utilizza il metodo `delete` del `DocumentClient` per eliminare un utente dal database;

Parametri:

```
* username: String
```

Username dell'utente che si vuole rimuovere dal sistema;

```
+ updateUser(user: User): ErrorObservable
```

Implementazione del metodo dell'interfaccia `UsersDAO`. Utilizza il metodo `update` del `DocumentClient` per aggiornare i dati relativi ad un utente presente all'interno del database;

Parametri:

```
* user: User
```

Parametro contenente i dati relativi all'utente che si vuole modificare;

```
+ <<Create>> createUsersDAODynamoDB(db : AWS::DynamoDB): UsersDAODynamoDB
```

Constructor della classe `UsersDAODynamoDB`. Permette di effettuare la dependency injection di `AWS::DynamoDB`;

Parametri:

```
* db : AWS::DynamoDB
```

Parametro contenente un riferimento al modulo di Node.js da utilizzare per l'accesso al database `DynamoDB` contenente la tabella degli utenti;

UserService

- **Nome:** `UserService`;
- **Tipo:** `Class`;
- **Descrizione:** questa classe si occupa di realizzare il microservizio `Auth` e, tramite `UsersDAO`, di interagire con il database degli utenti registrati;
- **Utilizzo:** fornisce i metodi che implementano le lambda function necessarie alla gestione degli utenti. Questa classe non interagisce direttamente con il database, ma fa utilizzo di `UsersDAO`, il quale nasconde i meccanismi di accesso e persistenza dei dati nel database;
- **Attributi:**
 - `users: UsersDAO`
Attributo che permette di contattare `UsersDAO`, il quale fornisce i meccanismi d'accesso al database degli utenti registrati;
- **Metodi:**

```
+ getUserList(event: LambdaEvent, context: LambdaContext): void
```

Metodo che implementa la lambda function che si occupa di restituire l'array degli utenti registrati;

Parametri:

```
* event: LambdaEvent
```

Parametro che rappresenta la richiesta ricevuta dal `VocalAPI`. Il campo `body` di questo attributo conterrà una stringa vuota;

```
* context: LambdaContext
```

Parametro utilizzato dalle lambda function per inviare la risposta. Il `body` del `LambdaResponse`, ottenuto dal metodo `LambdaContext::succeed`, conterrà un Array di oggetti di tipo `User`;

```
+ addUser(event: LambdaEvent, context: LambdaContext): void
```

Metodo che implementa la lambda function che si occupa di aggiungere un utente registrato;

Parametri:

```
* event: LambdaEvent
```

Parametro contenente, all'interno del campo **body** sotto forma di stringa in formato JSON, un oggetto **User** contenente tutti i dati relativi ad un utente da inserire;

```
* context: LambdaContext
```

Parametro utilizzato dalle lambda function per inviare la risposta. La risposta, contenuta nel **LambdaResponse** parametro del metodo **LambdaContext::succeed**, possiede un attributo **body**, il quale conterrà una stringa vuota. Il risultato delle operazioni di questo metodo sarà deducibile tramite il valore dell'attributo **LambdaResponse::statusCode**;

```
+ removeUser(event: LambdaIdEvent, context: LambdaContext): void
```

Metodo che implementa la lambda function che si occupa di rimuovere un utente registrato;

Parametri:

```
* event: LambdaIdEvent
```

Parametro contenente, all'interno del campo **pathParameters**, lo username dell'utente registrato che si vuole eliminare;

```
* context: LambdaContext
```

Parametro utilizzato dalle lambda function per inviare la risposta. Il **body** del **LambdaResponse**, parametro del metodo **LambdaContext::succeed**, conterrà una stringa vuota e il risultato di questa operazione sarà deducibile dal valore dell'attributo **LambdaResponse::statusCode**;

```
+ updateUser(event: LambdaIdEvent, context: LambdaContext): void
```

Metodo che implementa la lambda function che si occupa di aggiornare i dati di un utente registrato;

Parametri:

```
* event: LambdaIdEvent
```

Parametro contenente all'interno del campo **body**, sotto forma di stringa in formato JSON, un oggetto di tipo **User** contenente i dati da aggiornare e, all'interno del campo **pathParameters**, lo username dell'utente da modificare;

```
* context: LambdaContext
```

Parametro utilizzato dalle lambda function per inviare la risposta. Il **body** del **LambdaResponse**, parametro del metodo **LambdaContext::succeed**, conterrà una stringa vuota e il risultato di questa operazione sarà deducibile dal valore dell'attributo **LambdaResponse::statusCode**;

```
+ getUser(event: LambdaIdEvent, context: LambdaContext): void
```

Metodo che implementa la lambda function che si occupa di restituire i dati relativi ad un utente a partire dal suo username;

Parametri:

```
* event: LambdaIdEvent
```

Parametro contenente, all'interno del campo **pathParameters**, lo username dell'utente registrato del quale si vogliono ottenere i dati;

```
* context: LambdaContext
```

Parametro utilizzato dalle lambda function per inviare la risposta. Il **body** del **LambdaResponse**, parametro del metodo **LambdaContext::succeed**, conterrà un oggetto, sotto forma di stringa in formato JSON, di tipo **User**, contenente i dati relativi all'utente ritornato;

```
+ <<create>> createUsersService(user: UserDAO): UsersService
```

Metodo che permette di creare uno `UsersService`. Permette la dependency injection avente come oggetto uno `UsersDAO`;

Parametri:

```
* user: UserDAO
```

Attributo contenente lo `UsersDAO`;

VocalLoginModule

- **Nome:** `VocalLoginModule`;
- **Tipo:** `Class`;
- **Descrizione:** questa classe si occupa di realizzare e raggruppare tutte le operazioni necessarie all'identificazione di un utente registrato, tramite il microservizio di Speaker Recognition (SR). ;
- **Utilizzo:** fornisce un meccanismo per creare, eliminare ed ottenere un utente registrato associato ad un `Enrollment`, il quale è necessario all'identificazione di un utente tramite il microservizio di Speak Recognition (SR). Permette quindi di associare i parametri relativi ad un `Enrollment` ad un utente registrato.
È soggetta ad una constructor-based dependency injection, la quale ha come oggetto un `VocalLoginModuleConfig`;
- **Attributi:**
 - `min_confidence: int`
Attributo contenente il grado di confidenza minimo accettabile nel confronto tra ciò che l'utente comunica, al fine di effettuare l'accesso come utente registrato, e quella che dovrebbe essere la sua impronta vocale precedentemente costruita tramite il meccanismo di `Enrollment`;
- **Metodi:**

```
+ createUser(): String
```

Metodo che permette di creare un `User`;

```
+ <<Create>> createVocalLoginModule(conf: VocalLoginModuleConfig): VocalLoginModule
```

Metodo che permette di costruire un `VocalLoginModule`. Permette la dependency injection che ha come oggetto un `VocalLoginModuleConfig`;

Parametri:

```
* conf: VocalLoginModuleConfig
```

Parametro attraverso il quale viene passata la configurazione di `VocalLoginModule`;

```
+ addEnrollment(id: String, audio: Blob): Error
```

Metodo che permette di aggiungere un `Enrollment`;

Parametri:

```
* id: String
```

Parametro relativo a ???;

```
* audio: Blob
```

Parametro contenente l'audio relativo alla frase di riconoscimento pronunciata;

```
+ deleteUser(id: String): Error
```

Metodo che permette di eliminare un `User` a partire da un `id`;

Parametri:

```
* id: String
```

Parametro relativo a ??;


```
+ getList(): SRUserArray
Metodo che ritorna una lista di SRUser;

+ getUser(id: String): SRUser
Metodo che ritorna un SRUser a partire da un id;
Parametri:

    * id: String
        Parametro contenente l'identificativo dello User;

+ resetEnrollments(id: String): Error
Metodo che permette di resettare un enrollment a partire da un id;
Parametri:

    * id: String
        Parametro relativo a ???;

+ doLogin(id: String, audio: Blob): Error
Metodo che permette di effettuare il login a partire da un id e da un audio di identificazione;
Parametri:

    * id: String
        Parametro contenente l'identificativo ???;

    * audio: Blob
        Parametro contenente l'audio relativo alla frase di riconoscimento pronunciata;
```

Back-end::Notifications

Package che contiene le componenti relative al microservizio che si occupa delle notifiche.

Classi

Action

- **Nome:** Action;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare gli attributi relativi ad una Action come descritto nelle API di Slack. Rappresenta un button in un messaggio Slack;
- **Utilizzo:** fornisce gli attributi di una Action. La classe `Attachment` ne contiene un array. Per la relativa documentazione, consultare la pagina <https://api.slack.com/docs/message-buttons>;
- **Attributi:**
 - + **name:** String
Attributo contenente il nome dell'azione. Se ci sono più azioni con lo stesso nome, solo una di esse può essere in uno stato attivato;
 - + **text:** String
Attributo contenente il testo del bottone dell'azione;
 - + **style:** String[0..1]
Attributo che definisce lo stile del bottone. Per una lista degli stili disponibili e una loro descrizione fare riferimento alla documentazione di Slack (https://api.slack.com/docs/message-buttons#action_fields);

- + **type:** `String`
Attributo contenente il tipo dell'azione. Al momento l'unico valore accettato è "button". Fare riferimento alle API di Slack per informazioni aggiornate (https://api.slack.com/docs/message-buttons#action_fields). ;
- + **value:** `String[0..1]`
Attributo contenente il valore dell'azione. Se sono presenti diverse azioni con lo stesso nome, può essere utilizzato per distinguere diversi intenti;
- + **confirm:** `ConfirmationFields[0..1]`
Attributo contenente i dati relativi al dialogo di conferma, che nel caso di bottoni con azioni che possono avere effetti particolarmente "distruttivi" permette di chiedere un'ulteriore conferma prima di compiere effettivamente tali azioni;

Attachment

- **Nome:** `Attachment`;
- **Tipo:** `Class`;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare i dati relativi ad un `Attachment`. ;
- **Utilizzo:** fornisce gli attributi degli `Attachment` che dovranno essere aggiunti ad un `NotificationMessage`. Un `Attachment`, contenuto in un `NotificationsMessage`, permette di aggiungere del significato ad un `NotificationMessage` ed arricchirlo tramite immagini, colori ed altro. Per la relativa documentazione, consultare questa pagina <https://api.slack.com/docs/message-buttons>;
- **Attributi:**
 - + **title:** `String[0..1]`
Attributo contenente il titolo dell'attachment;
 - + **fallback:** `String`
Attributo contenente un messaggio mostrato agli utenti che utilizzano un'interfaccia che non supporta gli attachments;
 - + **callback_id:** `String`
Attributo contenente l'id della collezione di bottoni all'interno dell'attachment;
 - + **color:** `String[0..1]`
Attributo contenente il colore dell'attachment;
 - + **actions:** `ActionArray`
Attributo contenente una array di `Action` da includere nell'attachment. Questo array può contenere al massimo cinque `Action`;

ConfirmationFields

- **Nome:** `ConfirmationFields`;
- **Tipo:** `Class`;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare gli attributi relativi ad un messaggio di conferma di Slack;
- **Utilizzo:** fornisce gli attributi per rappresentare un messaggio di conferma di Slack. Per la relativa documentazione, consultare la pagina <https://api.slack.com/docs/message-buttons>;
- **Attributi:**

+ **title:** `String[0..1]`
Attributo contenente il titolo della finestra di pop up;

+ **text:** `String`
Attributo contenente la descrizione dettagliata delle conseguenze della relativa **Action** e contestualizza le scelte fornite dal button;

+ **ok.text:** `String[0..1]`
Attributo contenente il testo del bottone che serve per confermare la relativa **Action**. Il valore di default per questo attributo è "Okay". ;

+ **dismiss.text:** `String[0..1]`
Attributo contenente il testo del bottone che serve per cancellare la relativa **Action**. Il valore di default per questo attributo è "Cancel". ;

NotificationChannel

- **Nome:** `NotificationChannel`;
- **Tipo:** `Class`;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare i parametri relativi al canale Slack nel quale spedire il `NotificationMessage`;
- **Utilizzo:** fornisce gli attributi di un `NotificationChannel`.
Per consultare la relativa documentazione, consultare questa pagina <https://api.slack.com/types/channel>;
- **Attributi:**
 - + **id:** `String`
Attributo contenente l'id del canale Slack;
 - + **name:** `String`
Attributo contenente il nome del canale Slack;
 - + **created:** `String`
Attributo contenente l'unix timestamp relativo a creator;
 - + **creator:** `String`
Attributo contenente l'id dell'utente Slack che ha creato il canale Slack;
 - + **is_archived:** `boolean`
Attributo contenente un valore che permette di capire se un canale è stato archiviato o meno. Un canale è archiviato se non è più parte delle conversazioni attive;
 - + **is_member:** `boolean`
Attributo contenente un valore booleano che permette di capire se il membro chiamante fa parte del canale Slack;
 - + **num_members:** `int`
Attributo contenente il numero dei partecipanti al canale Slack;
 - + **topic:** `Topic`
Attributo contenente l'argomento di discussione del canale Slack;
 - + **purpose:** `Purpose`
Attributo contenente lo scopo per il quale il canale Slack è stato creato;

NotificationMessage

- **Nome:** `NotificationMessage`;

- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare i parametri relativi al messaggio da spedire nel NotificationChannel;
- **Utilizzo:** fornisce gli attributi di un NotificationChannel.
Per consultare la relativa documentazione, consultare questa pagina <https://api.slack.com/docs/message-formatting>;
- **Attributi:**
 - + **text:** String
Attributo contenente il testo del NotificationMessage;
 - + **attachments:** AttachmentArray[0..1]
Attributo contenente l'array degli attachments del NotificationMessage;
 - + **response_type:** String[0..1]
Attributo contenente il modo con il quale notificare. Questo attributo può assumere uno tra i seguenti valori:
 - * **in_channel**, che mostra il NotificationMessage ai membri del canale con il message button già cliccato;
 - * **ephemeral**, che mostra il NotificationMessage ai membri del canale che cliccano sul message button.

Per la relativa documentazione, consultare questa pagina <https://api.slack.com/docs/message-buttons>;

NotificationService

- **Nome:** NotificationService;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di realizzare il microservizio Notifications;
- **Utilizzo:** fornisce i metodi che implementano le lambda function necessarie per notificare la persona desiderata sul relativo canale Slack;
- **Attributi:**
 - **BOT_TOKEN:** String
Attributo utilizzato per autenticarsi come bot;
- **Metodi:**
 - + **getChannelList(event: LambdaEvent, context: LambdaContext): void**
Metodo che implementa la lambda function che si occupa di restituire l'array dei canali Slack disponibili. ;
Parametri:
 - * **event:** LambdaEvent
Parametro che rappresenta la richiesta ricevuta dal VocalAPI. Il campo body di questo attributo conterrà una stringa vuota;
 - * **context:** LambdaContext
Parametro utilizzato dalle lambda function per inviare la risposta. Il body del LambdaResponse, parametro del metodo LambdaContext::succeed, conterrà un Array di oggetti di tipo NotificationChannel;
 - + **sendMsg(event: LambdaEvent, context: LambdaContext): Msg**
Metodo che implementa la lambda function che si occupa di inviare il messaggio alla

persona desiderata;

Parametri:

* **event:** `LambdaEvent`

Parametro contenente, all'interno del campo `body` sotto forma di stringa in formato JSON, un oggetto contenente tutti i dati relativi ad un messaggio da inviare. Tali dati sono:

```
1 {
2     "msg": "NotificationMessage",
3     "send_to": "String"
4 }
```

Dove `msg` è un oggetto di tipo `NotificationMessage`, mentre `send_to` è una stringa contenente il mittente del messaggio;

* **context:** `LambdaContext`

Parametro utilizzato dalle lambda function per inviare la risposta. La risposta, contenuta nel `LambdaResponse` parametro del metodo `LambdaContext::succeed`, possiede un attributo `body`, il quale conterrà una stringa vuota. Il risultato delle operazioni di questo metodo sarà deducibile tramite il valore dell'attributo `LambdaResponse::statusCode`;

Purpose

- **Nome:** Purpose;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare i dati relativi al Purpose di un canale Slack. ;
- **Utilizzo:** fornisce un meccanismo per impostare e ottenere i valori dei parametri relativi ad un Purpose. Il purpose di un canale Slack è lo scopo per il quale è stato creato. Per la relativa documentazione, consultare la seguente pagina <https://api.slack.com/methods/channels.setPurpose>;
- **Attributi:**
 - + **value:** `String`
Attributo contenente il valore del Purpose;
 - + **creator:** `String`
Attributo contenente l'id del creatore dello scopo del canale Slack;
 - + **last_set:** `String`
Attributo contenente un unix timestamp relativo all'ultima modifica;

Topic

- **Nome:** Topic;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare i dati relativi al Topic di un canale Slack;
- **Utilizzo:** fornisce gli attributi del topic di un canale Slack. Il Topic di un canale Slack è l'oggetto o tema delle discussioni in esso. Per la relativa documentazione, consultare questa pagina <https://api.slack.com/methods/channels.setTopic>;
- **Attributi:**

```
+ value: String
Attributo contenente il valore del Topic;

+ creator: String
Attributo contenente l'id del creatore del topic del canale Slack;

+ last_set: String
Attributo contenente un unix timestamp relativo all'ultima modifica;
```

Back-end::Rules

Package che contiene le componenti necessarie alla gestione delle impostazioni (direttive) dell'assistente virtuale.

Classi

RulesDAO

- **Nome:** RulesDAO;
- **Tipo:** Interface;
- **Descrizione:** questa classe si occupa di astrarre le modalità di accesso al database per questo microservizio, contenente le Rule;
- **Utilizzo:** fornisce a **RuleService** un meccanismo per accedere al database, contenente le Rule, senza conoscerne le modalità di implementazioni e di persistenza del database. A partire da un identificativo, permette operazioni di lettura, scrittura e rimozione delle Rule;
- **Attributi:**
 - DB: AWS::DynamoDB
Attributo che permette di contattare il database contenente le direttive;
- **Metodi:**
 - + addRule(rule: Rule): ErrorObservable
Metodo che permette di aggiungere una Rule al database. L'Observable restituito non riceverà alcun valore, ma verrà completato in caso di aggiunta della Rule avvenuta con successo. In caso di errore durante l'aggiunta della Rule, gli Observer interessati verranno notificati tramite la chiamata del loro metodo error() con i dati relativi all'errore verificatosi;
Parametri:
 - * rule: Rule
Parametro contenente la Rule da aggiungere;
 - + getRulesList(): RuleObservable
L'Observable restituito manderà agli Observer le direttive ottenute, una alla volta, e poi chiama il loro metodo complete. Nel caso in cui si verifichi un errore, gli Observer iscritti verranno notificati tramite la chiamate del loro metodo error con i dati relativi all'errore verificatosi;
 - + removeRule(id: int): ErrorObservable
Metodo che permette di rimuovere una Rule dal database. L'Observable restituito non riceverà alcun valore, ma verrà completato in caso di rimozione della Rule avvenuta con successo. In caso di errore durante la rimozione della Rule, gli Observer interessati verranno notificati tramite la chiamata del loro metodo error() con i dati relativi

all'errore verificatosi;

Parametri:

* **id: int**

Parametro contenente l'id della Rule;

+ **getRule(id: int): RuleObservable**

Metodo che permette di ottenere una Rule a partire dal suo Id. L'**Observable** restituito riceverà l'oggetto rappresentante tale Rule, e verrà completato. Nel caso in cui la Rule richiesta non sia presente nel database, gli **Observer** interessati non riceveranno alcun valore, ma verranno notificati tramite la chiamata del loro metodo **error()**;

Parametri:

* **id: int**

Parametro contenente l'id della Rule da recuperare;

+ **updateRule(rule: Rule): ErrorObservable**

Metodo che permette di aggiornare una Rule. L'**Observable** restituito non riceverà alcun valore, ma verrà completato in caso di aggiornamento della Rule avvenuta con successo. In caso di errore durante l'aggiornamento della Rule, gli **Observer** interessati verranno notificati tramite la chiamata del loro metodo **error()** con i dati relativi all'errore verificatosi;

Parametri:

* **rule: Rule**

Parametro contenente la Rule;

TasksDAO

- **Nome:** TasksDAO;
- **Tipo:** Interface;
- **Descrizione:** questa classe si occupa di astrarre le modalità d'accesso al database per questo microservizio, contenente i Task;
- **Utilizzo:** fornisce a **RuleService** un meccanismo per accedere al database, contenente i compiti da applicare a certeRule, senza conoscerne le modalità di implementazioni e di persistenza del database. A partire da un identificativo, permette operazioni di lettura, scrittura e rimozione dei Task;
- **Metodi:**

+ **getTaskList(): TaskObservable**

Metodo che permette di ottenere la lista dei compiti. L'**Observable** restituito manderà agli **Observer** i compiti ottenuti, una alla volta, e poi chiama il loro metodo **complete**. Nel caso in cui si verifichi un errore, gli **Observer** iscritti verranno notificati tramite la chiamate del loro metodo **error** con i dati relativi all'errore verificatosi;

+ **addTask(task: Task): ErrorObservable**

Metodo che permette di aggiungere un Task al database.

L'**Observable** restituito non riceverà alcun valore, ma verrà completato in caso di aggiunta della funzione avvenuta con successo. In caso di errore durante l'aggiunta del Task, gli **Observer** interessati verranno notificati tramite la chiamata del loro metodo **error()** con i dati relativi all'errore verificatosi;

Parametri:

* **task: Task**

Parametro contenente il compito;

```
+ removeTask(id: int): ErrorObservable
```

Metodo che permette di rimuovere un compito dal DB a partire dal suo id.

L'**Observable** restituito non riceverà alcun valore, ma verrà completato in caso di rimozione del **Task** avvenuta con successo. In caso di errore durante la rimozione del **Task**, gli **Observer** interessati verranno notificati tramite la chiamata del loro metodo **error()** con i dati relativi all'errore verificatosi;

Parametri:

```
* id: int
```

Parametro contenente l'id della funzione;

```
+ getTask(id: int): ErrorObservable
```

Metodo che permette di ottenere una funzione a partire dal suo id.

L'**Observable** restituito riceverà l'oggetto rappresentante tale **Function**, e verrà completato. Nel caso in cui la funzione richiesta non sia presente nel database, gli **Observer** interessati non riceveranno alcun valore, ma verranno notificati tramite la chiamata del loro metodo **error()**;

Parametri:

```
* id: int
```

Parametro contenente l'id della funzione;

```
+ updateTask(task: Task): ErrorObservable
```

Metodo che permette di aggiornare un compito. L'**Observable** restituito non riceverà alcun valore, ma verrà completato in caso di aggiunta del **Task** con successo. In caso di errore durante l'aggiunta del **Task**, gli **Observer** interessati verranno notificati tramite la chiamata del loro metodo **error()** con i dati relativi all'errore verificatosi;

Parametri:

```
* task: Task
```

Parametro contenente il compito;

Rule

- **Nome:** Rule;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare i dati relativi ad una Rule, ovvero una direttiva definita da un amministratore;
- **Utilizzo:** fornisce i metodi getter e setter per i parametri relativi ad una direttiva, i quali dovranno essere memorizzati nel database per questo microservizio. Tramite il metodo **setTask**, è soggetta ad una setter-based dependency injection che ha come oggetto una **RuleTaskInstance**.
È utilizzata dalla classe **RulesDAO** e dalle classi che utilizzano quest'ultima;

- **Attributi:**

```
- targets: RuleTargetArray
```

Attributo contenente i targets della Rule;

```
- name: String
```

Attributo contenente il nome della Rule;

```
- id: int
```

Attributo contenente l'id della Rule;

```
- ac_mode: int
```

Attributo contenente 0 per positivo, 1 per negativo;

- `ac_list: StringArray`
Attributo contenente l'array di stringhe di id amministratori abilitati/disabilitati in base a valore di `ac_mode`;
- `enabled: boolean`
Attributo contenente un valore che dice se la Rule è abilitata o meno;
- `task: RuleTaskInstance`
Attributo contenente il compito della Rule;

- **Metodi:**

+ `<<Create>> Rule(targ: RuleTargetArray, name: String, ac_mode: int, ac_list: StringArray, en: boolean, task: String): Rule`

Metodo che permette di instanziare un oggetto `Rule` a partire da un nome, una lista dei targets un `ac_mode`, un `ac_list`, un compito da applicare e un valore booleano per abilitarla o meno;

Parametri:

- * `targ: RuleTargetArray`
Parametro contenente l'array dei targets da assegnare alla Rule;
- * `name: String`
Parametro contenente il nome da assegnare alla Rule;
- * `ac_mode: int`
Parametro contenente l'`ac_mode` da assegnare alla Rule;
- * `ac_list: StringArray`
Parametro contenente l'array di id degli amministratori abilitati/disabilitati da assegnare alla Rule;
- * `en: boolean`
Parametro contenente il valore booleano da assegnare alla Rule per abilitarla o meno;
- * `task: String`
Parametro contenente il compito da assegnare alla Rule;

+ `setTargets(targ: RuleTargetArray): void`

Metodo che permette di passare un Array contenente i targets per la Rule;

Parametri:

- * `targ: RuleTargetArray`
Parametro contenente l'array dei targets da passare;

+ `setName(name: String): void`

Metodo che permette di passare il nome per la Rule;

Parametri:

- * `name: String`
Parametro contenente il nome della Rule da passare;

+ `setId(id: int): void`

Metodo che permette di passare l'id per la Rule;

Parametri:

- * `id: int`
Parametro contenente l'id da passare;

+ `setAcMode(acm: int): void`

Metodo che permette di passare l'`ac_mode` per la Rule;

Parametri:

```
* acm: int
    Parametro contenente l'ac_mode da passare;

+ setAcList(ac1: StringArray): void
    Metodo che permette di passare gli id degli amministratori abilitati/disabilitati per la Rule;
    Parametri:

    * ac1: StringArray
        Parametro contenente l'ac_list da passare;

+ setEnabled(en: boolean): void
    Metodo che permette di passare un valore da impostare ad enabled per la Rule;
    Parametri:

    * en: boolean
        Parametro contenente il valore booleano da passare;

+ setTarget(function: RuleTaskInstance): void
    Metodo che permette di passare il compito da applicare per la Rule;
    Parametri:

    * function: RuleTaskInstance
        Parametro contenente la function da passare;

+ getName(): String
    Metodo che permette di ottenere il nome della Rule;

+ getTargets(): RuleTargetArray
    Metodo che permette di ottenere l'array contenente i targets della Rule;

+ getId(): int
    Metodo che permette di ottenere l'id della Rule;

+ getAcMode(): int
    Metodo che permette di ottenere l'ac_mode della Rule;

+ getAcList(): StringArray
    Metodo che permette di ottenere la lista degli id degli amministratori abilitati/disabilitati della Rule;

+ isEnabled(): boolean
    Metodo che permette di capire se la Rule è abilitata o meno;

+ getTask(): void
    Metodo che permette di ottenere la Task applicata dalla Rule;
```

RulesDAODynamoDB

- **Nome:** RulesDAODynamoDB;
- **Tipo:** Class;
- **Descrizione:** classe che si occupa di implementare l'interfaccia RulesDAO, utilizzando un database DynamoDB come supporto per la memorizzazione dei dati;

- **Utilizzo:** implementa i metodi dell'interfaccia `RulesDAO` interrogando un database DynamoDB. Utilizza `AWS::DynamoDB::DocumentClient` per l'accesso al database. La dependency injection dell'oggetto `AWS::DynamoDB` viene fatta utilizzando il costruttore;
- **Attributi:**
 - `db: AWS::DynamoDB`
Attributo contenente un riferimento al modulo di Node.js utilizzato per l'accesso al database DynamoDB contenente la tabella degli utenti;
- **Metodi:**
 - + `addRule(rule: Rule): ErrorObservable`
Implementazione del metodo definito nell'interfaccia `RulesDAO`. Utilizza il metodo `put` del `DocumentClient` per aggiungere la Rule al database;
Parametri:
 - * `rule: Rule`
Parametro contenente la Rule da aggiungere;
 - + `getRule(id: String): RuleObservable`
Implementazione del metodo definito nell'interfaccia `RulesDAO`. Utilizza il metodo `get` del `DocumentClient` per ottenere i dati relativi ad uno `Rule` dal database;
Parametri:
 - * `id: String`
Parametro contenente l'id della Rule da recuperare;
 - + `getRuleList(): RuleObservable`
Implementazione del metodo dell'interfaccia `RulesDAO`. Utilizza il metodo `scan` del `DocumentClient` per ottenere la lista delle Rule dal database;
 - + `removeRule(id: String): ErrorObservable`
Implementazione del metodo dell'interfaccia `RulesDAO`. Utilizza il metodo `delete` del `DocumentClient` per eliminare una Rule dal database;
Parametri:
 - * `id: String`
Parametro contenente l'id della Rule;
 - + `updateRule(rule: Rule): ErrorObservable`
Implementazione del metodo dell'interfaccia `RulesDAO`. Utilizza il metodo `update` del `DocumentClient` per aggiornare i dati relativi ad una Rule presente all'interno del database;
Parametri:
 - * `rule: Rule`
Parametro contenente la Rule da aggiornare;
 - + `<> createRulesDAODynamoDB(db: AWS::DynamoDB): RulesDAODynamoDB`
Constructor della classe `RulesDAODynamoDB`. Permette di effettuare la dependency injection di `AWS::DynamoDB`;
Parametri:
 - * `db: AWS::DynamoDB`
Parametro contenente un riferimento al modulo di Node.js da utilizzare per l'accesso al database DynamoDB contenente la tabella delle rule;

RulesService

- **Nome:** `RulesService`;

- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di realizzare il microservizio **Rules** e di interagire con il database delle direttive;
- **Utilizzo:** fornisce i metodi che implementano le lambda function necessarie alla gestione delle **Rule** e relative **Task**. Questa classe non interagisce direttamente con il database, ma fa utilizzo di **TasksDAO** e **RulesDAO**, le quali nascondono i meccanismi di accesso e persistenza dei dati nel database;
- **Attributi:**
 - **rules:** **RulesDAO**
Attributo che permette di contattare il **RulesDAO**, il quale permette l'accesso al database delle **Rule**;
 - **task:** **TasksDAO**
Attributo che permette di contattare il **TasksDAO**, il quale permette di accedere al database delle **Task**;
- **Metodi:**
 - + **addRule(event: LambdaEvent, context: LambdaContext): void**
Metodo che implementa la lambda function che si occupa di aggiungere una **Rule**;
Parametri:
 - * **event:** **LambdaEvent**
Parametro contenente, all'interno del campo **body** sotto forma di stringa in formato JSON, un oggetto **Rule** contenente tutti i dati relativi ad una **Rule** da inserire;
 - * **context:** **LambdaContext**
Parametro utilizzato dalle lambda function per inviare la risposta. Il **body** del **LambdaResponse**, parametro del metodo **LambdaContext::succeed**, conterrà una stringa vuota e il risultato di questa operazione sarà deducibile dal valore dell'attributo di **LambdaResponse::statusCode**;
 - + **deleteRule(event: LambdaIdEvent, context: LambdaContext): void**
Metodo che implementa la lambda function che si occupa di eliminare una **Rule**;
Parametri:
 - * **event:** **LambdaIdEvent**
Parametro contenente, all'interno del campo **pathParameters**, l'identificativo della **Rule** che si vuole eliminare;
 - * **context:** **LambdaContext**
Parametro utilizzato dalle lambda function per inviare la risposta. Il **body** del **LambdaResponse**, ottenuto dal metodo **LambdaContext::succeed**, conterrà una stringa vuota e il risultato di questa operazione sarà deducibile dal valore dell'attributo **LambdaResponse::statusCode**;
 - + **updateRule(event: LambdaIdEvent, context: LambdaContext): void**
Metodo che implementa la lambda function che si occupa di aggiornare una **Rule**;
Parametri:
 - * **event:** **LambdaIdEvent**
Parametro contenente all'interno del campo **body**, sotto forma di stringa in formato JSON, un oggetto di tipo **Rule** contenente i dati da aggiornare e, all'interno del campo **pathParameters**, l'identificativo della **Rule** da modificare;
 - * **context:** **LambdaContext**
Parametro utilizzato dalle lambda function per inviare la risposta. Il **body** del **LambdaResponse**, ottenuto dal metodo **LambdaContext::succeed**, conterrà una

stringa vuota e il risultato di questa operazione sarà deducibile dal valore dell'attributo `LambdaResponse::statusCode`;

+ `getRule(event: LambdaIdEvent, context: LambdaContext): void`

Metodo che implementa la lambda function che si occupa di ottenere una `Rule`;

Parametri:

* `event: LambdaIdEvent`

Parametro contenente, all'interno del campo `pathParameters`, l'identificativo della `Rule` della quale si vogliono ottenere i dati;

* `context: LambdaContext`

Parametro utilizzato dalle lambda function per inviare la risposta. Il `body` del `LambdaResponse`, ottenuto dal metodo `LambdaContext::succeed`, conterrà, sotto forma di stringa in formato JSON, un oggetto di tipo `Rule`, contenente i dati relativi alla `Rule` ritornata;

+ `getRuleList(event: LambdaEvent, context: LambdaContext): void`

Metodo che implementa la lambda function che si occupa di ottenere l'array delle `Rule`;

Parametri:

* `event: LambdaEvent`

Parametro contenente, all'interno del campo `body`, una stringa vuota;

* `context: LambdaContext`

Parametro utilizzato dalle lambda function per inviare la risposta. Il `body` del `LambdaResponse`, parametro del metodo `LambdaContext::succeed`, conterrà, sotto forma di una stringa in formato JSON, l'array delle `Rule` disponibili;

+ `getTaskList(event: LambdaEvent, context: LambdaContext): void`

Metodo che implementa la lambda function che si occupa di ottenere l'array delle `Task` disponibili;

Parametri:

* `event: LambdaEvent`

Parametro contenente, all'interno del campo `body`, una stringa vuota;

* `context: LambdaContext`

Parametro utilizzato dalle lambda function per inviare la risposta. Il `body` del `LambdaResponse`, ottenuto dal metodo `LambdaContext::succeed`, conterrà, sotto forma di una stringa in formato JSON, un Array di oggetti di tipo `Function`;

+ `getTask(event: LambdaIdEvent, context: LambdaContext): void`

Metodo che implementa la lambda function che si occupa di ottenere una `Task`;

Parametri:

* `event: LambdaIdEvent`

Parametro contenente, all'interno del campo `pathParameters`, l'identificativo della `Rule` della quale si vuole ottenere la `Task`;

* `context: LambdaContext`

Parametro utilizzato dalle lambda function per inviare la risposta. Il `body` del `LambdaResponse`, parametro del metodo `LambdaContext::succeed`, conterrà, sotto forma di una stringa in formato JSON, un oggetto di tipo `RuleTaskInstance`, contenente i dati relativi alla `Task` ritornata;

+ `<<Create>> createRulesService(task: TasksDAO, rules: RulesDAO): RulesService`

Metodo che permette di creare un `RulesService`. Permette la dependency injection avente come oggetti un `RulesDAO` e `TasksDAO`;

Parametri:

* `task: TasksDAO`

Attributo contenente il `TasksDAO`;

```
* rules: RulesDAO
    Attributo contenente il RulesDAO;
```

RuleTarget

- **Nome:** RuleTarget;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare i dati relativi a un target di una Rule, ovvero la persona alla quale è indirizzata quest'ultima;
- **Utilizzo:** fornisce gli attributi di un target di una Rule che dovranno essere memorizzati nel database per questo microservizio. Viene utilizzata dalla classe Rule per definire un Array di Target ai quali applicare la sua funzione;
- **Attributi:**
 - + company: String
Attributo contenente il nome dell'azienda;
 - + name: String
Attributo contenente il nome del target;
 - + member: String
Attributo contenente ??? perchè un target dovrebbe contenere il nome di uno di zero12?;

RuleTaskInstance

- **Nome:** RuleTaskInstance;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare i dati relativi al compito di una direttiva;
- **Utilizzo:** fornisce un meccanismo per specificare la modifica di comportamento del sistema in seguito all'applicazione di una determinata direttiva. ;
- **Attributi:**
 - + task: String
Attributo contenente il task da applicare;
 - + params: Array
Attributo contenente l'array dei parametri relativi alla funzione;

Task

- **Nome:** Task;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare la funzione che dovrà essere applicata ad una Rule;
- **Utilizzo:** fornisce gli attributi delle funzioni (ovvero dei compiti che una certa Rule svolge) che dovranno essere applicate a certe Rule, i quali dovranno essere memorizzati nel database per questo microservizio;
- **Attributi:**

```
+ function: String
Attributo contenente la funzione da applicare;

+ id: int
Attributo contenente l'id della funzione;
```

TasksDAODynamoDB

- **Nome:** TasksDAODynamoDB;
- **Tipo:** Class;
- **Descrizione:** classe che si occupa di implementare l'interfaccia **TasksDAO**, utilizzando un database **DynamoDB** come supporto per la memorizzazione dei dati;
- **Utilizzo:** implementa i metodi dell'interfaccia **TasksDAO** interrogando un database **DynamoDB**. Utilizza **AWS::DynamoDB::DocumentClient** per l'accesso al database. La dependency injection dell'oggetto **AWS::DynamoDB** viene fatta utilizzando il costruttore;
- **Attributi:**
 - **db:** **AWS::DynamoDB**
Attributo contenente un riferimento al modulo di Node.js utilizzato per l'accesso al database **DynamoDB** contenente la tabella degli utenti;
- **Metodi:**
 - + **addFunction(fun: Function): ErrorObservable**
Implementazione del metodo definito nell'interfaccia **FunctionsDAO**. Utilizza il metodo **put** del **DocumentClient** per aggiungere la funzione al database;
Parametri:
 - * **fun: Function**
Parametro contenente la funzione;
 - + **getFunction(id: String): TaskObservable**
Implementazione del metodo definito nell'interfaccia **FunctionsDAO**. Utilizza il metodo **get** del **DocumentClient** per ottenere i dati relativi ad una **Function** dal database;
Parametri:
 - * **id: String**
Parametro contenente l'id della funzione;
 - + **getFunctionList(): TaskObservable**
Implementazione del metodo dell'interfaccia **FunctionsDAO**. Utilizza il metodo **scan** del **DocumentClient** per ottenere la lista delle funzioni dal database;
 - + **removeFunction(id: String): ErrorObservable**
Implementazione del metodo dell'interfaccia **FunctionsDAO**. Utilizza il metodo **delete** del **DocumentClient** per eliminare una funzione dal database;
Parametri:
 - * **id: String**
Parametro contenente l'id della funzione;
 - + **updateFunction(fun: Function): ErrorObservable**
Implementazione del metodo dell'interfaccia **FunctionsDAO**. Utilizza il metodo **update** del **DocumentClient** per aggiornare i dati relativi ad una funzione presente all'interno del database;
Parametri:

```

* fun: Function
    Parametro contenente la funzione;

+ <> createFunctionsDAO DynamoDB(db: AWS::DynamoDB): TasksDAO DynamoDB
    Constructor della classe FunctionsDAO DynamoDB. Permette di effettuare la dependency
    injection di AWS::DynamoDB;
    Parametri:

* db: AWS::DynamoDB
    Parametro contenente un riferimento al modulo di Node.js da utilizzare per l'accesso
    al database DynamoDB contenente la tabella delle funzioni;

```

Back-end::STT

Package che include le classi che si occupano di fornire le funzionalità di Speech to text.

Classi

STTModule

- **Nome:** STTModule;
- **Tipo:** Interface;
- **Descrizione:** classe che definisce l'interfaccia dei moduli utilizzati per le operazioni di Speech-To-Text (STT);
- **Utilizzo:** fornisce l'interfaccia che deve essere implementata dai moduli che permettono l'accesso alle funzionalità di STT;
- **Metodi:**

```

+ speechToText(audio: Buffer, type: String): StringPromise
    Questo metodo permette di ricavare in modo asincrono il testo da un file audio. Viene
    utilizzato da VirtualAssistantAPI, il quale invierà il testo ottenuto dall'invocazione
    di questo metodo all'assistente virtuale. Restituisce una promise che verrà soddisfatta
    con una stringa contenente il testo estratto;
    Parametri:

* audio: Buffer
    Buffer contenente l'audio da cui si vuole estrarre il testo;

* type: String
    Parametro contenente la descrizione del formato in cui i dati sono memorizzati in
    audio;

```

STTWatsonAdapter

- **Nome:** STTWatsonAdapter;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di convertire l'interfaccia fornita dal servizio di Speech to Text di IBM in una più adatta alle esigenze dell'applicazione, definita da STTModule. Facendo da Adapter tra le API del servizio di Speech to Text di IBM (adaptee) e l'interfaccia STTModule (target) utilizzata da APIGateway::VocalAPI, permette l'interoperabilità tra queste due interfacce;

- **Utilizzo:** Fornisce a `STTModule` un meccanismo che permette di interrogare le API del servizio Watson Speech to Text di IBM, in modo da consentire a `APIGateway::VocalAPI` l'utilizzo di quest'ultime utilizzando un'interfaccia distinta e più consona alle proprie esigenze. È soggetta ad una constructor-based dependency injection, la quale ha come oggetti:

- uno `StreamBufferModule` contenente ... ??? ;
- uno `SpeechToTextV1` ???

Per ulteriori informazioni, consultare le documentazioni presenti alle seguenti pagine:

- <https://www.ibm.com/watson/developercloud/doc/speech-to-text/>;
- <https://github.com/watson-developer-cloud/node-sdk#speech-to-text>.

;

- **Attributi:**

- `stt: SpeechToTextV1`

Attributo contenente il `SpeechToTextV1Module`, creato a partire dai parametri `name` e `password` forniti al costruttore. Viene utilizzato per estrarre il contenuto testuale di un file audio utilizzando il servizio Watson Speech To Text di IBM;

- `stream_buffer: StreamBufferModule`

Attributo contenente il `StreamBufferModule` di cui è stata effettuata la dependency injection nel costruttore. Viene utilizzata per creare un `ReadableStream` di node a partire da un buffer;

- **Metodi:**

```
+ <<Create>> createSTTWatsonAdapter(sb: StreamBufferModule, stt: SpeechToTextV1): STTWatsonAdapter
```

Costruttore di `STTWatsonAdapter`, che permette di effettuare la dependency injection di `StreamBufferModule` e di `SpeechToTextV1`;

Parametri:

- * `sb: StreamBufferModule`

Parametro tramite il quale si effettua la dependency injection di `StreamBufferModule`;

- * `stt: SpeechToTextV1`

Parametro tramite il quale viene effettuata la dependency injection di `SpeechToTextV1`.

;

```
+ speechToText(audio: Buffer, type: String): StringPromise
```

Implementa il metodo `speechToText` contenuto nell'interfaccia;

Parametri:

- * `audio: Buffer`

Parametro contenente l'audio dal quale si vuole estrarre il testo;

- * `type: String`

Parametro contenente il formato in cui sono memorizzati i dati all'interno di `audio`;

Back-end::Utility

Package contenente classi e interfacce, dallo scopo generico, utili ad altri package del back-end.

Classi

Error

- **Nome:** Error;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare i dati relativi ad un errore che può avvenire;
- **Utilizzo:** fornisce gli attributi necessari a descrivere gli errori che si possono verificare. L'attributo `code` contiene un valore uguale a 0 nel caso non si sia verificato nessun errore, diverso da 0 altrimenti. Nel caso di `code` diverso da 0, l'attributo `msg` contiene una stringa che descrive l'errore verificatosi;
- **Attributi:**
 - + `code: String`
Attributo contenente il codice dell'errore;
 - + `msg: String[0..1]`
Attributo contenente il messaggio dell'errore;

LambdaContext

- **Nome:** LambdaContext;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare un oggetto context passato alle lambda function;
- **Utilizzo:** fornisce un metodo che permette di inviare una risposta all'API Gateway. ;
- **Metodi:**
 - + `succeed(res: LambdaResponse): void`
Metodo che permette di inviare una risposta all'API Gateway;
Parametri:
 - * `res: LambdaResponse`
Attributo contenente la risposta da inviare;
 - + `getRemainingTimeInMillis(): int`
DA TOGLIERE DA TOGLIERE DA TOGLIERE DA TOGLIEREDA TOGLIERE
DA TOGLIEREDA TOGLIERE DA TOGLIEREDA TOGLIERE DA TOGLIERE-
DA TOGLIERE DA TOGLIEREDA TOGLIERE DA TOGLIEREDA TOGLIERE DA
TOGLIEREDA TOGLIERE DA TOGLIEREDA TOGLIERE DA TOGLIEREDA TO-
GLIERE DA TOGLIEREDA TOGLIERE DA TOGLIEREDA TOGLIERE DA TO-
GLIERE;

LambdaEvent

- **Nome:** LambdaEvent;
- **Tipo:** Class;
- **Descrizione:** classe che rappresenta l'oggetto event che viene passato alle LambdaFuction dall'API Gateway con l'integrazione Lambda Proxy;

- **Utilizzo:** fornisce i parametri necessari alla lambda function per gestire le richieste che arrivano all'API Gateway;
- **Figlio:** LambdaIdEvent;
- **Attributi:**
 - + **body:** `String`
Stringa contenente i dati ricevuti dall'API Gateway. Le singole Lambda Function si dovranno occupare dell'interpretazione di tale stringa nel formato adeguato (ad esempio JSON, testo, ecc.);
 - + **headers:** `StringAssocArray`
Array associativo di stringhe contenente gli headers HTTP della richiesta ricevuta dall'API Gateway;

LambdaIdEvent

- **Nome:** LambdaIdEvent;
- **Tipo:** Class;
- **Descrizione:** classe che rappresenta un oggetto event che viene passato alle Lambda function dall'API Gateway con l'integrazione Lambda Proxy;
- **Utilizzo:** eredita da LambdaEvent ed aggiunge l'attributo pathParameters;
- **Padre:** LambdaEvent;
- **Attributi:**
 - + **pathParameters:** `PathIdParam`
Parametro contenente l'id dell'utente del quale si vogliono ottenere i dati;

LambdaResponse

- **Nome:** LambdaResponse;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare la risposta di una lambda function;
- **Utilizzo:** fornisce alle lambda function gli attributi necessari per inviare una risposta ad API Gateway;
- **Attributi:**
 - + **statusCode:** `int`
Attributo contenente il codice di stato HTTP che dovrà avere la risposta;
 - + **headers:** `StringAssocArray`
Attributo contenente l'array associativo nel quale la chiave indica il nome di un header HTTP da mandare nella risposta ed il valore è una stringa contenente il valore di tale header;
 - + **body:** `String`
Attributo contenente il corpo della risposta;

PathIdParam

- **Nome:** PathIdParam;
- **Tipo:** Class;

- **Descrizione:** questa classe rappresenta i parametri path di una richiesta caratterizzata da un unico parametro che è un identificativo;
- **Utilizzo:** fornisce l'attributo che contiene l'identificativo della risorsa richiesta;
- **Attributi:**
 - + `id: String`
Attributo contenente l'id della risorsa richiesta;

ProcessingResult

- **Nome:** `ProcessingResult`;
- **Tipo:** `Class`;
- **Descrizione:** questa classe si occupa di rappresentare il campo result della risposta fornita da api.ai;
- **Utilizzo:** fornisce gli attributi per rappresentare l'oggetto result relativo ad una risposta di api.ai.
Per la relativa documentazione, consultare la pagina <https://docs.api.ai/docs/query#response>;
- **Attributi:**
 - + `source: String`
Attributo contenente la sorgente dalla quale è stata ricavata la risposta. ;
 - + `resolvedQuery: String`
Attributo contenente il testo dell'intents utilizzato per interrogare api.ai;
 - + `action: String`
Attributo contenente l'azione da eseguire;
 - + `actionIncomplete: boolean`
Attributo contenente un valore booleano che indica se i parametri necessari a far eseguire l'azione sono stati forniti tutti o meno. ;
 - + `parameters: StringAssocArray`
Attributo contenente un oggetto costituito dai parametri necessari per portare a termine l'azione;
 - + `contexts: ContextArray`
Attributo contenente l'array dei context attivi;
 - + `fulfilment: Fulfillment`
Attributo contenente i dati ricevuti dal webhook;
 - + `score: Real`
Attributo contenente un numero, contenuto nell'intervallo [0,1], che indica la sicurezza con la quale è stato trovato il relativo intent;
 - + `metadata: Metadata`
Attributo contenente dati relativi a intents e contexts;

StatusObject

- **Nome:** `StatusObject`;
- **Tipo:** `Class`;
- **Descrizione:** questa classe si occupa di rappresentare lo status object della richiesta mandata ad api.ai, che indica se quest'ultima ha avuto successo o meno. ;

- **Utilizzo:** fornisce gli attributi per rappresentare l'oggetto status object relativo ad una richiesta mandata ad api.ai.
Per la relativa documentazione, consultare la pagina <https://docs.api.ai/docs/status-object>;
- **Attributi:**
 - + **code:** `int`
Attributo contenente il codice di stato HTTP;
 - + **errorType:** `String`
Attributo contenente una breve descrizione dell'errore verificatosi. In caso non se ne verificano, questo attributo avrà valore pari a "success";
 - + **errorId:** `String`
Attributo contenente l'id dell'errore verificatosi;
 - + **errorDetails:** `String`
Attributo contenente la descrizione dettagliata dell'errore verificatosi. In caso non se ne verificano, questo attributo non sarà ritornato;

Back-end::VirtualAssistant

Package contenente le componenti del microservizio dell'assistente virtuale.

Classi

AgentsDAO

- **Nome:** `AgentsDAO`;
- **Tipo:** `Interface`;
- **Descrizione:** questa classe si occupa di astrarre le modalità di accesso al database contenente gli `Agent` disponibili;
- **Utilizzo:** fornisce a `WebhookService` un meccanismo per accedere ai dati relativi agli `Agent`, senza conoscerne le modalità di implementazioni e di persistenza del database. A partire da un identificativo, permette operazioni di lettura, scrittura e rimozione degli `Agent`;
- **Attributi:**
 - **db:** `AWS::DynamoDB`
Da fare;
- **Metodi:**
 - + **getAgentsList():** `AgentObservable`
L'`Observable` restituito manderà agli `Observer` gli agents ottenuti, uno alla volta, e poi chiama il loro metodo `complete`. Nel caso in cui si verifichi un errore, gli `Observer` iscritti verranno notificati tramite la chiamata del loro metodo `error` con i dati relativi all'errore verificatosi;
 - + **updateAgent(agent: Agent):** `ErrorObservable`
Metodo che permette di aggiornare un `Agent`. L'`Observable` restituito non riceverà alcun valore, ma verrà completato in caso di aggiornamento dell'`Agent` avvenuta con successo. In caso di errore durante l'aggiornamento dell'`Agent`, gli `Observer` interessati verranno notificati tramite la chiamata del loro metodo `error()` con i dati relativi all'errore verificatosi;
Parametri:

```

* agent: Agent
    Parametro contenente l'agente da aggiornare;

+ removeAgent(name: String): ErrorObservable
    Metodo che permette di rimuovere un Agent a partire da un name. L'Observable
    restituito non riceverà alcun valore, ma verrà completato in caso di rimozione dell'Agent
    avvenuta con successo. In caso di errore durante la rimozione dell'Agent, gli Observer
    interessati verranno notificati tramite la chiamata del loro metodo error() con i dati
    relativi all'errore verificatosi;
    Parametri:

    * name: String
        Parametro contenente il name dell'agente da rimuovere;

+ addAgent(agent: Agent): ErrorObservable
    Metodo che permette di aggiungere un Agent. L'Observable restituito non riceverà
    alcun valore, ma verrà completato in caso di aggiunta dell'Agent avvenuta con succes-
    so. In caso di errore durante l'aggiunta dell'Agent, gli Observer interessati verranno
    notificati tramite la chiamata del loro metodo error() con i dati relativi all'errore ve-
    rificatosi;
    Parametri:

    * agent: Agent
        Parametro contenente l'agente da aggiungere al database;

+ getAgent(name: String): AgentObservable
    Metodo che ritorna un Agent a partire da un name. L'Observable restituito riceverà
    l'oggetto rappresentante tale Agent, e verrà completato. Nel caso in cui l'Agent richiesto
    non sia presente nel database, gli Observer interessati non riceveranno alcun valore, ma
    verranno notificati tramite la chiamata del loro metodo error();
    Parametri:

    * name: String
        Parametro contenente il name dell'agente da ricevere;

```

VAModule

- **Nome:** VAModule;
- **Tipo:** Interface;
- **Descrizione:** questa classe definisce l'interfaccia dei moduli che si occupano di interrogare le API di un assistente virtuale;
- **Utilizzo:** fornisce la definizione dei metodi che dovranno essere implementati, in maniera tale da permettere ad un modulo che si interfacci con api.ai di essere utilizzato da VAService;
- **Figlio:** ApiAiVAAdapter;
- **Metodi:**

```

+ query(str: VAQuery): VResponse
    Metodo che permette di interrogare l'assistente virtuale;
    Parametri:

    * str: VAQuery
        Parametro contenente i dati necessari all'interrogazione dell'assistente virtuale;

```

WebhookService

- **Nome:** WebhookService;

- **Tipo:** Interface;
- **Descrizione:** questa classe definisce l'interfaccia dei moduli che implementano dei webhook conformi alle specifiche di api.ai;
- **Utilizzo:** fornisce una serie di metodi necessari per definire un servizio che soddisfi i requisiti per webhook di api.ai. Per la relativa documentazione, consultare la pagina <https://docs.api.ai/docs/webhook>;
- **Figli:** ConversationWebhookService, AdministrationWebhookService;
- **Metodi:**

+ `webhook(event: LambdaEvent, context: LambdaContext): void`

Metodo che fornisce l'interfaccia di una lambda function compatibile con i requisiti per webhook di api.ai;

Parametri:

* `event: LambdaEvent`

Parametro contenente i dati mandati da api.ai al WebhookService;

* `context: LambdaContext`

Parametro utilizzato dalle lambda function per inviare la risposta dal WebhookService ad api.ai. Il body del `LambdaResponse`, parametro del metodo `LambdaContext::succeed`, conterrà un oggetto, sotto forma di stringa in formato JSON, contenente i dati relativi all'utente ritornato.

Tali dati sono così organizzati:

```

1 {
2   "contextOut": "ContextArray",
3   "data": "Object",
4   "displayText": "String",
5   "followupEvent": "Object",
6   "source": "String",
7   "speech": "String"
8 }
```

Dove:

- `contextOut`: attributo contenente l'array dei context che la richiesta ha attivato;
- `data`: attributo contenente i dati che saranno inviati al client. Tali dati non sono processati da api.ai e quindi saranno nella forma originale;
- `displayText`: attributo contenente il testo che verrà mostrato sullo schermo dell'utente;
- `followupEvent`: attributo contenente parametri opzionali che il servizio web utilizzato vuole inviare ad api.ai;
- `source`: attributo contenente la risorse dei dati forniti come risposta;
- `speech`: attributo contenente la risposta testuale alla richiesta.

Per la relativa documentazione, consultare la pagina [Per una descrizione dettagliata si rimanda alla pagina https://docs.api.ai/docs/webhook#section-format-of-response-from-t](https://docs.api.ai/docs/webhook#section-format-of-response-from-t);

Agent

- **Nome:** Agent;

- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare i dati relativi ad un Agent di api.ai;
- **Utilizzo:** fornisce gli attributi di un Agent che dovranno essere memorizzati nel database per questo microservizio. In api.ai, un Agent ha lo scopo di trasformare il linguaggio naturale, ricevuto in input, in dati capibili per le applicazioni. Per la relativa documentazione, consultare questa pagina <https://docs.api.ai/docs/concept-agents>;
- **Attributi:**
 - + **name:** String
Nome dell'applicazione a cui è collegato l'agent. Per ogni applicazione, abbiamo un Agent;
 - + **token:** String
Attributo contenente il valore del token associato. Un agent è identificabile tramite esso;
 - + **lang:** String
Attributo contenente la lingua, la quale dovrà essere sempre inglese (en);

AgentsDAODynamoDB

- **Nome:** AgentsDAODynamoDB;
- **Tipo:** Class;
- **Descrizione:** classe che si occupa di implementare l'interfaccia **AgentsDAO**, utilizzando un database DynamoDB come supporto per la memorizzazione dei dati;
- **Utilizzo:** implementa i metodi dell'interfaccia **AgentsDAO** interrogando un database DynamoDB. Utilizza **AWS::DynamoDB::DocumentClient** per l'accesso al database. La dependency injection dell'oggetto **AWS::DynamoDB** viene fatta utilizzando il costruttore;
- **Attributi:**
 - **db:** **AWS::DynamoDB**
Attributo contenente un riferimento al modulo di Node.js utilizzato per l'accesso al database DynamoDB contenente la tabella degli utenti;
- **Metodi:**
 - + **addAgent(agent: Agent): ErrorObservable**
Implementazione del metodo definito nell'interfaccia **AgentsDAO**. Utilizza il metodo **put** del **DocumentClient** per aggiungere l'utente al database;
Parametri:
 - * **agent:** Agent
Parametro contenente l'agente da aggiungere al database;
 - + **getAgent(name: String): AgentObservable**
Implementazione del metodo definito nell'interfaccia **AgentsDAO**. Utilizza il metodo **get** del **DocumentClient** per ottenere i dati relativi ad uno **User** dal database;
Parametri:
 - * **name:** String
Parametro contenente il nome dell'agente da ricevere;
 - + **getAgentList(): AgentObservable**
Implementazione del metodo dell'interfaccia **AgentsDAO**. Utilizza il metodo **scan** del **DocumentClient** per ottenere la lista degli utenti dal database;


```
+ removeAgent(name: String): ErrorObservable
```

Implementazione del metodo dell'interfaccia **AgentsDAO**. Utilizza il metodo delete del **DocumentClient** per eliminare un utente dal database;

Parametri:

```
* name: String
```

Parametro contenente il nome dell'agente da rimuovere;

```
+ updateAgent(agent: Agent): ErrorObservable
```

Implementazione del metodo dell'interfaccia **AgentsDAO**. Utilizza il metodo update del **DocumentClient** per aggiornare i dati relativi ad un utente presente all'interno del database;

Parametri:

```
* agent: Agent
```

Parametro contenente l'agente da aggiornare;

```
+ <> createAgentsDAODynamoDB(db: AWS::DynamoDB): AgentsDAODynamoDB
```

Constructor della classe **AgentsDAODynamoDB**. Permette di effettuare la dependency injection di **AWS::DynamoDB**;

Parametri:

```
* db: AWS::DynamoDB
```

Parametro contenente un riferimento al modulo di Node.js da utilizzare per l'accesso al database **DynamoDB** contenente la tabella degli agenti;

ApiAiVAAdapter

- **Nome:** **ApiAiVAAdapter**;

- **Tipo:** **Class**;

- **Descrizione:** questa classe si occupa di convertire l'interfaccia fornita da **api.ai** in una più adatta alle esigenze dell'applicazione, definita da **VAModule**.

Facendo da Adapter tra le API di **api.ai** (adaptee) e l'interfaccia **VAModule** (target) utilizzata da **VAService**, permette l'interoperabilità tra queste due interfacce;

- **Utilizzo:** fornisce a **VAModule** un meccanismo che permette di interrogare le API di **api.ai**, in modo da consentire a **VAService** l'utilizzo di quest'ultime utilizzando un'interfaccia distinta e più consona alle proprie esigenze.

Per fare le richieste HTTP alle API, il metodo **query** utilizza il modulo **request-promise**. È soggetta ad una constructor-based dependency injection, la quale ha come oggetto un **Agent** relativo alle richieste da inviare. ;

- **Padre:** <<interface>> **VAModule**;

- **Attributi:**

```
- agent: Agent
```

Attributo contenente lo **Agent** al quale inviare le richieste;

```
- VERSION: String
```

Da fare;

- **Metodi:**

```
+ <<Create>> createApiAiVAAdapter(agent: Agent): ApiAiVAAdapter
```

Da fare;

Parametri:

```
* agent: Agent
```

Da fare;

```
+ query(str: VAQuery): VResponse
Metodo che permette di interrogare lo Agent in api.ai;
Parametri:

    * str: VAQuery
        Attributo contenente i dati relativi all'interrogazione da porre allo Agent in api.ai;

+ intents():
Metodo che permette;

+ contexts():
Metodo che permette;

+ userEntities():
Metodo che permette;
```

ButtonObject

- **Nome:** ButtonObject;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare l'oggetto buttons di api.ai;
- **Utilizzo:** fornisce gli attributi per rappresentare l'oggetto buttons relativo ad una risposta.
Per la relativa documentazione, consultare la pagina <https://docs.api.ai/docs/rich-messages>;
- **Attributi:**
 - + text: String
Attributo contenente il testo del button;
 - + postback: String
Attributo contenente il testo da inviare, ad api.ai o un URL, dopo che il button è stato selezionato;

Context

- **Nome:** Context;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare gli attributi relativi ad un Context in api.ai;
- **Utilizzo:** fornisce gli attributi di un Context. In api.ai, un Context è una stringa che rappresenta il contesto corrente nel quale un utente somministra una richiesta, in maniera tale da disambiguare il più possibile quest'ultime.
Per la relativa documentazione, consultare questa pagina <https://docs.api.ai/docs/concept-contexts>;
- **Attributi:**
 - + name: String
Attributo contenente il nome del Context;
 - + parameters: StringAssocArray
Attributo contenente l'array dei parametri che compongono il Context;

+ **lifespan:** int

Attributo contenente il lifespan del Context. Questo valore indica per quanti altri matched intents bisogna mantenere il Context. Per chiarire quanto appena detto, consultare la relativa documentazione in questa pagina <https://api.ai/blog/2015/11/23/Contexts/>;

Fulfillment

- **Nome:** Fulfillment;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare gli attributi relativi ad un fulfillment di api.ai;
- **Utilizzo:** fornisce gli attributi di un fulfillment. In api.ai un fulfillment è tutto ciò che viene ritornato dalla richiesta di utente. In particolare, è l'adempimento ad una richiesta e può comprendere diversi campi dati. Per chiarire quanto appena detto, consultare la relativa documentazione <https://docs.api.ai/docs/webhook>. ;
- **Attributi:**
 - + **speech:** String
Attributo contenente il testo relativo alla risposta fornita;
 - + **messages:** MsgObjectArray
Attributo contenente l'array di tutti i campi dati presenti nel messages. Il tipo è un MsgObject in quanto i tipi degli elementi di messages sono eterogenei fra loro;

Metadata

- **Nome:** Metadata;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare i dati relativi a intents e contexts;
- **Utilizzo:** fornisce gli attributi relativi ai dati di intents e contexts e viene utilizzato come attributo della classe **ProcessingResult**. Per la relativa documentazione, consultare la pagina <https://docs.api.ai/docs/query#response>;
- **Attributi:**
 - + **intentId:** String
Attributo contenente l'identificativo per un intent;
 - + **webhookUsed:** String
Attributo contenente un valore booleano che indica se è stato usato un webhook. Per chiarire quanto detto, consultare la relativa documentazione <https://docs.api.ai/docs/webhook>;
 - + **webhookForSlotFillingUsed:** String
Attributo contenente un valore booleano che indica se è stato usato un webhook for slot filling. Per chiarire quanto detto, consultare la relativa documentazione <https://docs.api.ai/docs/webhook#webhook-for-slot-filling>;
 - + **intentName:** String
Attributo contenente il nome dell'intent;

MsgObject

- **Nome:** MsgObject;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare e organizzare gli attributi relativi ad un MsgObject, il quale definisce un tipo unico tra tutti i messaggi (dal contenuto eterogeneo) contenuti in un Fulfillment;
- **Utilizzo:** fornisce gli attributi di un MsgObject, il quale sarà contenuto in un Fulfillment. La classe Fulfillment fa utilizzo di un array di essi per adempire alla richiesta di un utente. Per la relativa documentazione, consultare la pagina <https://docs.api.ai/docs/rich-messages>;
- **Attributi:**
 - + type: int
Attributo contenente;
 - + speech: String[0..1]
Attributo contenente la risposta dell'assistente;
 - + imageUrl: String[0..1]
Attributo contenente;
 - + title: String[0..1]
Attributo contenente il titolo del messaggio;
 - + subtitle: String[0..1]
Attributo contenente il subtitle definito da api.ai;
 - + buttons: ButtonObjectArray[0..1]
Attributo contenente l'array di button definiti da api.ai;
 - + replies: StringArray[0..1]
Attributo contenente;
 - + payload: Object[0..1]
Attributo contenente il payload;

ResponseBody

- **Nome:** ResponseBody;
- **Tipo:** Class;
- **Descrizione:** questa classi si occupa di rappresentare il corpo della risposta, fornita dall'assistente virtuale, in seguito ad un'interrogazione;
- **Utilizzo:** fornisce gli attributi necessari per rappresentare il corpo della risposta dell'assistente virtuale.
Viene passata come parametro al metodo runCmd delle varie applicazioni, il quale si occupa di estrarre i parametri necessari ad eseguire il comando ricevuto;
- **Attributi:**
 - + text_request: String
Attributo contenente il testo della richiesta dell'utente;
 - + text_response: String
Attributo che contiene il testo della risposta dell'assistente virtuale;
 - + contexts: ObjectAssocArray[0..1]
Array contenente i context ricevuti dall'assistente virtuale;

+ **data:** Object[0..1]

Attributo che può essere utilizzato per scambiare dati tra client e l'eventuale webhook. Il suo contenuto dipende dal servizio webhook utilizzato. Il client si deve occupare di reinviare i dati presenti in questo campo di una determinata risposta nella richiesta successiva. In particolare, questo attributo viene utilizzato per lo scambio del token dello agent in api.ai;

VAEventObject

- **Nome:** VAEventObject;
- **Tipo:** Class;
- **Descrizione:** questa classe rappresenta il campo event di una richiesta all'assistente virtuale;
- **Utilizzo:** fornisce i campi necessari all'interrogazione di un assistente virtuale, utilizzando il nome di un evento e dei parametri al posto della stringa di query **VAQuery::text**. Per la relativa documentazione, consultare la pagina <https://docs.api.ai/docs/concept-events>;
- **Attributi:**

+ **name:** String

Nome dell'evento che si vuole far eseguire all'assistente virtuale;

+ **data:** Object[0..1]

Oggetto contenente i parametri necessari all'assistente virtuale per l'esecuzione dell'evento specificato. Tali parametri saranno inseriti come coppie chiave-valore, dove la chiave indica il nome del parametro;

VAQuery

- **Nome:** VAQuery;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare una richiesta da porre all'assistente virtuale;
- **Utilizzo:** fornisce gli attributi necessari che compongono una richiesta all'assistente virtuale. Viene utilizzata dal metodo **query** delle classi che implementano l'interfaccia **VAModule** per interrogare i rispettivi assistenti virtuali;
- **Attributi:**

+ **text:** String[0..1]

Attributo la cui presenza è obbligatoria a meno che non sia presente l'attributo event. Contiene il testo della frase che si vuole comunicare all'assistente;

+ **event:** VAEventObject[0..1]

Attributo la cui presenza è obbligatoria, a meno che non sia presente l'attributo text. Indica l'evento di cui si richiede l'esecuzione.

Per una definizione di evento fare riferimento alla documentazione di api.ai alla pagina <https://docs.api.ai/docs/concept-events>;

+ **session_id:** String

Attributo contenente l'id della sessione dell'assistente virtuale. Deve essere generato dal client;

+ **data:** Object[0..1]

Oggetto che permette l'invio dei dati necessari all'eventuale webhook per svolgere la sua funzione;

VAResponse

- **Nome:** VAResponse;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare l'intera risposta dell'assistente virtuale, fornita in seguito ad un'interrogazione ;
- **Utilizzo:** fornisce gli attributi relativi ad una risposta dell'assistente virtuale. Viene utilizzata dalle classi che implementano l'interfaccia **VAModule**, in maniera tale da fornire la risposta dell'assistente virtuale;
- **Attributi:**
 - + **session_id:** String
Rappresenta l'id univoco della sessione corrente dell'assistente virtuale;
 - + **res:** ResponseBody
Attributo contenente il corpo della risposta ricevuta dall'assistente virtuale. Il contenuto di questo attributo sarà passato senza modifiche al metodo **Client::ApplicationManager::Application::** dell'applicazione;
 - + **action:** String
Attributo contenente l'azione che l'assistente virtuale richiede di compiere al client. La stringa è nel formato **[nome_applicazione.comando]**, dove **nome_applicazione** indica l'applicazione che deve essere eseguita, mentre **comando** indica il comando che il client deve far eseguire all'applicazione.
Ad esempio, **conversation.displayMsgs** comunica al client di far eseguire il comando **displayMsgs** all'applicazione che si occupa di mostrare la conversazione. ;

VAService

- **Nome:** VAService;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare il microservizio Virtual Assistant. ;
- **Utilizzo:** fornisce il metodo necessario all'interrogazione dell'assistente virtuale;
- **Attributi:**
 - **agents:** AgentsDAO
Attributo che permette di contattare **AgentsDAO**, il quale fornisce i meccanismi d'accesso al database degli **Agent** disponibili;
 - **va_module:** VAModule
Attributo contenente il **VAModule**.
VAModule è un'interfaccia, implementata da **ApiAiVAAdapter**. I metodi invocati saranno quelli appartenenti a quest'ultima classe;
- **Metodi:**
 - + **<<Create>> createVAService(agents: AgentsDAO, va: VAModule): VAService**
Costruttore che realizza una dependency injection, avente come oggetti un **AgentsDAO** e un **VAModule**;
Parametri:
 - * **agents:** AgentsDAO
Da fare;
 - * **va:** VAModule
Da fare;

```
+ query(event: LambdaEvent, context: LambdaContext): void
```

Questo metodo implementa la lambda function che si occupa dell'interrogazione dell'assistente virtuale. A partire da una richiesta contenente il testo della richiesta dell'utente, questo metodo esegue una chiamata all'endpoint /query di api.ai e, a partire dalla risposta ricevuta, manda la risposta utilizzando il context;

Parametri:

* **event: LambdaEvent**

Parametro contenente l'evento con i dati relativi alla richiesta di API Gateway. Il campo **body** di questo evento conterrà una stringa in formato JSON nel formato seguente:

```
1 {
2     "app": "nome\_applicazione",
3     "query": "VAQuery"
4 }
```

con **app** stringa che corrisponde al nome dell'applicazione che manda la richiesta, e **query** oggetto del tipo **VAQuery** contenente i dati relativi alla query da mandare all'assistente virtuale;

* **context: LambdaContext**

Parametro utilizzato dalle lambda function per inviare la risposta. La risposta, contenuta nel **LambdaResponse** parametro del metodo **LambdaContext::succeed**, possiede un attributo **body**, il quale conterrà il corpo di essa sotto forma di una stringa in formato JSON, organizzando i dati nel seguente modo:

```
1 {
2     "action": "String",
3     "res":
4     {
5         "contexts": "ObjectAssocArray",
6         "data": "Object",
7         "text\_request": "String",
8         "text\_response": "String"
9     },
10    "session\_id": "String"
11 }
```

Dove:

- **action**: attributo contenente l'azione che l'assistente virtuale richiede di compiere al client. La stringa è nel formato **[nome_applicazione.comando]**, dove **nome_applicazione** indica l'applicazione che deve essere eseguita, mentre **comando** indica il comando che il client deve far eseguire all'applicazione. Ad esempio, **conversation.displayMsgs** comunica al client di far eseguire il comando **displayMsgs** all'applicazione che si occupa di mostrare la conversazione;
- **res**: attributo contenente il corpo della risposta ricevuta dall'assistente virtuale. Il contenuto di questo attributo sarà passato senza modifiche al metodo **Client::ApplicationManager::Application::runCmd** dell'applicazione. In questo oggetto abbiamo i campi **contexts** (array contenente i context ricevuti dall'assistente virtuale), **data** (attributo che può essere utilizzato per scambiare dati tra client e l'eventuale webhook. Il suo contenuto dipende dal servizio webhook utilizzato. Il client si deve occupare di reinviare i dati presenti in questo campo di una determinata risposta nella richiesta successiva. In particolare, questo attributo viene utilizzato per lo scambio, tra client e back-end,

del token dello agent in api.ai), `text_request` (attributo contenente il testo della richiesta dell'utente) e `text_response` (attributo che contiene il testo della risposta dell'assistente virtuale).

- `session_id`: attributo contenente l'id univoco della sessione corrente dell'assistente virtuale.

;

Client

Package che racchiude tutte le componenti del client. Il pattern utilizzato per organizzare le componenti è quello di un'architettura event-driven.

Classi

ConversationApp

- **Nome:** ConversationApp;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare l'applicazione di conversazione tra un utente e l'assistente virtuale;
- **Utilizzo:** fornisce un meccanismo, tramite i metodi e gli attributi forniti, per rappresentare e gestire la conversazione tra un utente e l'assistente virtuale;
- **Padre:** Application;
- **Metodi:**

- `onMsgReceived(msg: String): void`

Metodo che permette di creare l'HTMLElement del messaggio di risposta e di appenderlo all'interfaccia;

Parametri:

* `msg: String`

Parametro contenente il messaggio di risposta;

- `onMsgSent(msg: String): void`

Metodo che permette di creare l'HTMLElement del messaggio di richiesta e di appenderlo all'interfaccia;

Parametri:

* `msg: String`

Parametro contenente il messaggio di richiesta;

- `onDisplayMsgs(selfMsg: String, otherMsg: String): void`

Metodo che permette di appendere e visualizzare nell'interfaccia il messaggio di richiesta e il messaggio di risposta;

Parametri:

* `selfMsg: String`

Parametro contenente il messaggio di richiesta;

* `otherMsg: String`

Parametro contenente il messaggio di risposta;

- `onClear(): void`

Metodo che permette di ripulire l'interfaccia dai messaggi;


```
+ runCmd(cmd: String, params: ResponseBody): void
```

Metodo che permette di chiamare uno dei metodi privati, in base al comando ricevuto.

I comandi supportati sono:

- * `clear`: causa la chiamata del metodo `onClear`;

- * `msg`: causa la chiamata del metodo `onDisplayMsgs`, al quale vengono passati `params.text_request` e `params.text_response`.

```
;
```

Parametri:

- * `cmd`: `String`

Parametro contenente il comando da somministrare all'applicazione;

- * `params`: `ResponseBody`

Parametro contenente l'array dei parametri del comando da somministrare all'applicazione;

ObserverAdapter

- **Nome:** `ObserverAdapter`;

- **Tipo:** `Abstract Class`;

- **Descrizione:** questa classe astratta fornisce i metodi che permettono di mettere in pausa un `Observer`. Questa classe implementa l'interfaccia della libreria esterna `RxJS::Observer`;

- **Utilizzo:** implementa l'interfaccia `Observer`, definendo i metodi `next`, `complete`, `error` in modo che chiamino le rispettive funzioni di callback. Fornisce inoltre un meccanismo per mettere in pausa un `Observer`, in modo che ignori gli eventi che gli sono notificati. Nel caso in cui l'`Observer` sia messo in pausa, le funzioni di callback impostate non verranno chiamate;

- **Figli:** `PlayerObserver`, `ApplicationManagerObserver`, `LogicObserver`;

- **Attributi:**

- `paused`: `boolean`

Attributo che indica se l'`Observer` è in pausa;

- `next_cb`: `function(data : Object) : void`

Funzione di callback da chiamare quando viene chiamato il metodo `next`;

- `error_cb`: `function(err: Error): void`

Funzione di callback che viene chiamata quando viene chiamato il metodo `error`;

- `complete_cb`: `function(): void`

Funzione di callback che viene chiamata quando viene chiamato il metodo `complete`;

- **Metodi:**

```
+ next(data: Object): void
```

Implementazione del metodo dell'interfaccia. Si occupa di chiamare la funzione di callback impostata dalla chiamata al metodo `onNext`, passandole come parametro `data`;

Parametri:

- * `data`: `Object`

Parametro contenente i dati da passare a `next_cb`;

```
+ pause(): void
```

Metodo che permette di mettere in pausa l'`Observer`;

```

+ resume(): void
Metodo che permette di riavviare l'Observer;

+ <<Create>> createPausableObserver(): PausableObserver
Costruttore che imposta il valore di paused a false;

+ isPaused(): boolean
Metodo che permette di ottenere il valore di paused, in modo da capire se l'Observer è
in pausa o meno;

+ complete(): void
Implementazione del metodo dell'interfaccia. Si occupa di chiamare la funzione comple-
te_cb;

+ error(err: Error): void
Implementazione del metodo dell'interfaccia. Si occupa di chiamare la funzione di call-
back error_cb, passandole come parametro err;
Parametri:

    * err: Error
      Parametro che contiene i dati relativi all'errore verificatosi;

+ onNext(cb: function(data: Object): void): void
Metodo che permette di impostare la funzione di callback da chiamare quando viene
chiamato il metodo next;
Parametri:

    * cb: function(data: Object): void
      Funzione di callback;

+ onComplete(cb: function(): void): void
Metodo che permette di impostare la funzione di callback da chiamare quando viene
chiamato il metodo complete;
Parametri:

    * cb: function(): void
      Funzione di callback;

+ onError(cb: function(err: Error) : void): void
Metodo che permette di impostare la funzione di callback da chiamare quando viene
chiamato il metodo error;
Parametri:

    * cb: function(err: Error) : void
      Funzione di callback;

```

Client::ApplicationManager

Package contenente le classi che si occupano della gestione delle applicazioni con le quali l'utente può interagire.

Classi

ApplicationRegistryClient

- **Nome:** ApplicationRegistryClient;

- **Tipo:** Interface;
- **Descrizione:** interfaccia che fornisce i metodi che devono essere implementati dalle classi che si occupano di interrogare un registry della applicazioni;
- **Utilizzo:** fornisce al **Manager** l'interfaccia delle classi che si occuperanno di ottenere gli **ApplicationPackage**;
- **Metodi:**
 - + **register(name: String, pkg: ApplicationPackage): boolean**
Metodo che permette la registrazione di una applicazione nell'IApplicationRegistry-Client. Questo metodo deve però impedire l'inserimento di package "Parziali" di RegistryPackage, ovvero un package di tipo ApplicationPackage. ;
Parametri:
 - * **name: String**
Parametro contenente il nome sotto cui registrare il Package dell'applicazione;
 - * **pkg: ApplicationPackage**
Parametro contenente il Package che si desidera registrare nel registry;
 - + **query(name: String): ApplicationPackage**
Metodo che permette di ottenere il package di un'applicazione dal registry. Restituisce null in caso l'applicazione non sia disponibile;
Parametri:
 - * **name: String**
Parametro contenente il nome dell'applicazione che si vuole recuperare, come restituito dalle API;
- **Relazioni con le altre classi:**
 - IN **Manager**
 - IN **ApplicationRegistryLocalClient**
 - OUT **ApplicationPackage**

Application

- **Nome:** Application;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa della gestione dell'applicazione in esecuzione. È una classe astratta;
- **Utilizzo:** fornisce al client funzionalità per l'istanziamento dell'applicazione necessaria. ;
- **Figlio:** ConversationApp;
- **Attributi:**
 - **ui: HTMLElement**
Attributo contenente l'HTMLElement dell'interfaccia dell'applicazione. Per informazioni sul tipo HTMLElement, fare riferimento alla pagina seguente:<https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement>;
- **Metodi:**
 - + <<Create>> **createApplication(pkg: ApplicationPackage): Application**
Metodo che permette di costruire un Application a partire da un ApplicationPackage. Esegue il codice presente all'interno del campo **setup** di **pkg**, eseguendo il binding di **this** per far sì che tale codice abbia accesso all'oggetto che sta venendo creato, quindi crea il

metodo `runCmd` a partire dal codice presente all'interno di `pkg.cmdHandler`, eseguendo nuovamente il binding di `this`;

Parametri:

- * `pkg: ApplicationPackage`
Package contenente i dati relativi all'applicazione da istanziare;

+ `runCmd(cmd: String, params: ResponseBody): void`

Metodo che permette di eseguire un comando sull'applicazione. Viene creato a partire da `pkg`;

Parametri:

- * `cmd: String`
Parametro contenente il comando da somministrare all'applicazione;

- * `params: ResponseBody`
Parametro contenente l'array dei parametri del comando da somministrare all'applicazione;

+ `getUI(): HTMLElement`

Metodo che permette di restituire l'interfaccia dell'applicazione da eseguire, sotto forma di un oggetto di tipo `Element` (<https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement>). Questo `HTMLElement` verrà poi appeso al proprio frame da Manager;

- **Relazioni con le altre classi:**

- IN `State`
- OUT `ApplicationPackage`

ApplicationManagerObserver

- **Nome:** `ApplicationManagerObserver`;

- **Tipo:** `Class`;

- **Descrizione:** questa classe si occupa di inviare il testo contenente la risposta fornita dal sistema al Manager;

- **Utilizzo:** fornisce un meccanismo per eseguire una funzione all'arrivo di una notifica dal `DataArrivedObservable`;

- **Padre:** `ObserverAdapter`;

- **Metodi:**

- + `next(function(val: boolean): void): void`

Questo metodo, all'arrivo della notifica dal `DataArrivedSubject`, permette di passare all'`ApplicationManagerObserver` una funzione da eseguire.

;

Parametri:

- * `function(val: boolean): void`
Parametro contenente la funzione da eseguire;

- **Relazioni con le altre classi:**

- OUT `DataArrivedObservable`

ApplicationPackage

- **Nome:** ApplicationPackage;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare e definire il tipo di package che viene restituito da una interrogazione di un registry delle applicazioni;
- **Utilizzo:** fornisce gli attributi caratterizzanti una **Application**, necessari alla sua istanziazione. ;
- **Attributi:**
 - + name: String
Attributo contenente il nome dell'applicazione recuperata;
 - + setup: String
Attributo contenente il codice Javascript che deve essere eseguito nel costruttore dell'**Application**. Tale codice ha accesso al riferimento **this** dell'**Application**, e può utilizzarlo per definire i metodi e gli attributi dell'**Application** implementata;
 - + cmdHandler: String
Attributo contenente il codice Javascript del metodo **runCmd** dell'**Application** implementata. Tale codice ha accesso al riferimento **this** dell'**Application**;
 - + ui: String
Attributo contenente l'**Element** che racchiude l'interfaccia dell'applicazione;
- **Metodi:**
 - + <<Create>> createApplicationPackage(cmdHandler: String, name: String, setup: String, view: String): ApplicationPackage
Questo metodo permette di costruire un ApplicationPackage, a partire da un cmdHandler, name, setup e view di un Application presente nel LocalRegistry;
Parametri:
 - * cmdHandler: String
Parametro contenente l'url del cmdHandler, il quale si occupa di gestire i messaggi ricevuti dalle API. Questo attributo contiene codice JavaScript;
 - * name: String
Parametro contenente il nome dell'applicazione recuperata;
 - * setup: String
Parametro contenente il setup dell'applicazione in un dato istante;
 - * view: String
url;
- **Relazioni con le altre classi:**
 - IN **Application**
 - IN <<interface>> **ApplicationRegistryClient**

ApplicationRegistryLocalClient

- **Nome:** ApplicationRegistryLocalClient;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di implementare l'interfaccia fornita da **ApplicationRegistryClient**. Interroga un **LocalRegistry**;

- **Utilizzo:** fornisce i meccanismi necessari ad interrogare il `LocalRegistry` e ad aggiungere un `ApplicationPackage` al suo interno;
- **Attributi:**
 - `registry: ApplicationLocalRegistry`
Attributo che permette l'accesso all'`ApplicationLocalRegistry`;
- **Metodi:**
 - + `query(app_name: String): ApplicationPackage`
Metodo che permette di interrogare l'`IRegistryClient` a partire dal nome dell'applicazione, ottenendo l'`ApplicationPackage` relativa ad essa. ;
Parametri:
 - * `app_name: String`
Parametro contenente il nome dell'applicazione che si vuole recuperare;
 - + `register(name: String, pkg: ApplicationPackage): boolean`
Metodo relativo alla registrazione di una applicazione nell'`IRegistryClient`. Questo metodo deve però impedire l'inserimento di package "Parziali" di `RegistryPackage`, ovvero un package di tipo `ApplicationPackage`. ;
Parametri:
 - * `name: String`
Parametro contenente il nome dell'applicazione che si vuole registrare;
 - * `pkg: ApplicationPackage`
Parametro contenente l'`ApplicationPackage` relativo all'applicazione da registrare;
 - + `<<Create>> createApplicationRegistryLocalClient(client: ApplicationLocalRegistry): ApplicationRegistryLocalClient`
Costruttore che permette di effettuare una dependency injection di `ApplicationLocalRegistry`;
Parametri:
 - * `client: ApplicationLocalRegistry`
Parametro relativo all'`ApplicationLocalRegistry` di cui viene effettuata la dependency injection;
- **Relazioni con le altre classi:**
 - OUT `<<interface>> ApplicationRegistryClient`

Manager

- **Nome:** `Manager`;
- **Tipo:** `Class`;
- **Descrizione:** questa classe si occupa di gestire il cambio delle applicazioni nel client;
- **Utilizzo:** fornisce all'`ApplicationManager` un meccanismo per cambiare l'applicazione in esecuzione. Questo avviene salvando lo stato dell'applicazione corrente nello `State` e recuperando la nuova applicazione da `State` (se presente) o dal `ApplicationRegistryClient`;
- **Attributi:**
 - `registry_client: ApplicationRegistryClient`
Attributo utilizzato per ottenere i dati relativi ad un'applicazione da istanziare;
 - `state: State`
Attributo che tiene traccia delle applicazioni già eseguite e del loro stato;

- **frame:** `HTMLElement`

Attributo che contiene l'elemento del DOM al quale verrà appesa la view dell'applicazione. Per informazioni sul tipo `HTMLElement` fare riferimento alla pagina <https://developer.mozilla.org/en/docs/Web/API/HTMLElement>;

- **Metodi:**

+ **<<Create>> createManager(rc: ApplicationRegistryClient, frame: HTMLElement): Manager**

Metodo che permette di costruire un `Manager`. Permette di effettuare la dependency injection di un `ApplicationRegistryClient` e di un `Element` al manager, e si occupa di creare lo state iniziale;

Parametri:

* **rc: ApplicationRegistryClient**

Parametro che permette la dependency injection di un `ApplicationRegistryClient` all'interno di `Manager`;

* **frame: HTMLElement**

Parametro che permette la dependency injection di `HTMLElement` all'interno di `Manager`. Tale oggetto rappresenta l'elemento del DOM al quale verrà appesa l'interfaccia dell'applicazione;

+ **runApplication(app: String, cmd: String, params: ResponseBody): void**

Metodo che permette di passare a `Manager` il nome dell'applicazione e del comando da eseguire;

Parametri:

* **app: String**

Parametro contenente il nome dell'applicazione da eseguire;

* **cmd: String**

Parametro contenente il comando da dare all'applicazione `app`;

* **params: ResponseBody**

Parametro contenente l'insieme dei dati relativi alla risposta ricevuta;

+ **setFrame(frame: HTMLElement): void**

Metodo che permette di effettuare la dependency injection dell'elemento del DOM al quale sarà appesa l'interfaccia dell'applicazione da eseguire. Il nuovo `HTMLElement` sostituirà quello già presente all'interno di `Manager`, permettendo così di cambiare la posizione dell'interfaccia utente dell'applicazione all'interno della pagina;

Parametri:

* **frame: HTMLElement**

Parametro che contiene l'elemento del DOM;

- **Relazioni con le altre classi:**

- OUT `State`

- OUT **<<interface>>** `ApplicationRegistryClient`

State

- **Nome:** `State`;

- **Tipo:** `Class`;

- **Descrizione:** questa classe si occupa di salvare lo stato attuale delle applicazioni la cui esecuzione vuole essere sospesa;

- **Utilizzo:** fornisce al Manager un meccanismo per salvare lo stato attuale delle applicazioni la cui esecuzione è stata sospesa, in modo da poterlo recuperare una volta riattivata. Lo stato comprende tutti gli attributi e tutti i metodi dell'istanza dell'applicazione;
- **Attributi:**
 - apps: ApplicationAssocArray
Array associativo contenente le applicazioni che sono state sospese o in esecuzione. ;
- **Metodi:**
 - + addApp(app: Application, name: String): void
Metodo che permette di aggiungere un'applicazione allo State, sovrascrivendo quella già esistente se presente;
Parametri:
 - * app: Application
Parametro contenente l'Application da aggiungere allo State;
 - * name: String
Nome sotto il quale l'applicazione verrà salvata;
 - + getApp(name: String): Application
Metodo che permette di estrarre un'applicazione dallo State. Nel caso l'applicazione col nome specificato non sia presente, questo metodo restituisce null;
Parametri:
 - * name: String
Nome dell'applicazione della quale si vuole recuperare l'istanza in esecuzione;
- **Relazioni con le altre classi:**
 - IN Manager
 - OUT Application

Client::Logic

Package contenente le classi che gestiscono la logica del client e che si occupano della comunicazione con il back-end.

Classi

DataArrivedObservable

- **Nome:** DataArrivedObservable;
- **Tipo:** Class;
- **Descrizione:** classe che rappresenta un Observable che notifica gli observer interessati quando sono arrivati dei dati dall'API Gateway;
- **Utilizzo:** fornisce il meccanismo che permette agli observer di iscriversi per essere notificati quando arriva una risposta dall'API Gateway. Viene creata a partire dal DataArrivedSubject, per evitare di dover condividere quest'ultimo;
- **Relazioni con le altre classi:**
 - IN ApplicationManagerObserver
 - IN PlayerObserver
 - OUT DataArrivedSubject

DataArrivedSubject

- **Nome:** DataArrivedSubject;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di notificare gli observer collegati che i dati relativi ad una risposta sono arrivati;
- **Utilizzo:** fornisce un meccanismo che permette l'invio dei dati, il testo della risposta dell'assistente virtuale, una volta che questi ultimi sono stati forniti dal (sistema). Gli observer collegati sono:
 - ApplicationManagerObserver;
 - PlayerObserver.
- ;
- **Padre:** Subject;
- **Metodi:**
 - + next(val: boolean): void
Questo metodo permette di notificare il PlayerObserver e l'ApplicationManagerObserver;
Parametri:
 - * val: boolean
Parametro contenente il valore con il quale notificare il PlayerObserver e l'ApplicationManagerObserver;
 - + subscribe(obs: BoolObserver): Subscription
Questo metodo permette di iscrivere il PlayerObserver e l'ApplicationManagerObserver;
Parametri:
 - * obs: BoolObserver
Parametro contenente il valore con il quale il PlayerObserver e l'ApplicationManagerObserver devono essere iscritti;
- **Relazioni con le altre classi:**
 - IN DataArrivedObservable
 - IN Logic
 - OUT Logic

images/ClassHt

HttpError

- **Nome:** HttpError;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare un errore HTTP;
- **Utilizzo:** viene utilizzata per fornire un messaggio di errore in seguito a richieste HTTP fallite;
- **Attributi:**
 - + status: short
Attributo contenente il codice di stato HTTP, come definito nella sezione 6 dello standard rfc7231 (<https://tools.ietf.org/html/rfc7231#section-6>);

+ **statusText**: String
Attributo contenente un messaggio descrittivo dell'errore;

- **Relazioni con le altre classi:**

- IN `HttpPromise`
- IN `Logic`

HttpPromise

- **Nome:** `HttpPromise`;
- **Tipo:** `Class`;
- **Descrizione:** promise relativa ad una richiesta HTTP;
- **Utilizzo:** viene utilizzata per interfacciarsi alle richieste HTTP utilizzando l'approccio delle Promises al posto di quello dei callback fornito dalle API JavaScript. ;
- **Metodi:**

```
+ <<Create>> createHttpPromise(method: String, url: String, headers: StringAssocArray,
data: Object): HttpPromise
```

Constructor, si occupa di creare la promessa relativa ad una richiesta HTTP fatta con metodo, headers e url specificati;
Parametri:

- * **method**: String
Metodo della richiesta HTTP;
- * **url**: String
Url a cui fare la richiesta;
- * **headers**: StringAssocArray
Array associativo contenente gli headers. La chiave rappresenta il nome dell'header;
- * **data**: Object
Dati da mandare con la richiesta;

```
+ then(fullfillHandler: function(data : String), rejectHandler: function(error
: HttpError)):
```

Metodo utilizzato per specificare cosa fare nel caso in cui la Promise sia soddisfatta oppure rigettata;
Parametri:

- * **fullfillHandler**: function(data : String)
Metodo da chiamare in caso di promessa soddisfatta;
- * **rejectHandler**: function(error : HttpError)
Funzione da chiamare in caso di promessa rigettata;

- **Relazioni con le altre classi:**

- IN `Logic`
- IN `LogicObserver`
- OUT `HttpError`

Logic

- **Nome:** `Logic`;

- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di comunicare con l'API Gateway;
- **Utilizzo:** fornisce dei meccanismi per:
 - la comunicazione con l'API gateway;
 - pubblicare eventi all'arrivo di una risposta dall'API Gateway.
- ;
- **Attributi:**
 - **subject:** `DataArrivedSubject`
Attributo contenente il `DataArrivedSubject` per notificare il `DataArrivedObservable` che sono arrivati dei dati dall'API Gateway;
- **Metodi:**
 - + `sendData(audio: Blob): void`
Metodo che permette di inviare l'audio all'API Gateway;
Parametri:
 - * `audio: Blob`
Parametro contenente l'audio da inviare all'APIGateway;
 - + `<<Create>> createLogic(client: IEPRegistryClient): Logic`
Metodo che permette di istanziare Logic a partire da un IEPRegistryClient;
Parametri:
 - * `client: IEPRegistryClient`
Parametro contenente l'interfaccia per interrogare l'EndPointRegistryAdapter;
 - + `getObservable(): DataArrivedObservable`
Metodo utilizzato per ottenere un `Observable` per l'arrivo dei dati dall'API Gateway;
- **Relazioni con le altre classi:**
 - IN `DataArrivedSubject`
 - OUT `HttpError`
 - OUT `HttpPromise`
 - OUT `DataArrivedSubject`

LogicObserver

- **Nome:** `LogicObserver`;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di inviare l'audio relativo a ciò che l'utente ha comunicato a Logic. ;
- **Utilizzo:** fornisce un meccanismo per eseguire una funzione all'arrivo di una notifica da `SpeechEndObservable`;
- **Padre:** `ObserverAdapter`;
- **Metodi:**
 - + `next(function(val: boolean): void): void`
Questo metodo, all'arrivo della notifica dall'`EndSpeechDataSubject`, permette di passare

al LogicObserver un'operazione da eseguire. ;

Parametri:

```
* function(val: boolean): void
```

Parametro contenente la funzione da eseguire;

- **Relazioni con le altre classi:**

- OUT `SpeechEndObservable`

- OUT `HttpPromise`

Client::Recorder

Package contenente le classi che realizzano la registrazione audio.

Classi

Recorder

- **Nome:** Recorder;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa della registrazione della conversazione;
- **Utilizzo:** fornisce al client un meccanismo per registrare ciò che l'ospite comunica. Utilizza il volume rilevato per capire se l'ospite sta parlando, e genera i relativi comandi da mandare al `RecorderWorker`. Quando inizia una nuova registrazione, l'audio della registrazione precedente viene eliminato chiamando il comando clear del Worker;
- **Attributi:**
 - `threshold: real`
Attributo contenente la soglia di volume per la registrazione;
 - `max.silence: int`
Attributo contenente il tempo in ms atteso prima che venga fermata la registrazione a causa del silenzio (volume del suono registrato al di sotto di `threshold`);
 - `source: AudioNode`
Attributo contenente la sorgente audio dalla quale vogliamo registrare;
 - `worker: RecorderWorker`
Attributo contenente l'oggetto `RecorderWorker` sfruttato per la registrazione;
 - `max.listen: int`
Attributo contenente il tempo massimo di registrazione, espresso in ms;
 - `node: ScriptProcessorNode`
Attributo contenente il nodo utilizzato per la registrazione dell'audio. ;
- **Metodi:**
 - + `start(): void`
Metodo che mette il `Recorder` in ascolto, in attesa che la soglia minima di volume venga superata. Quando tale soglia viene superata, viene generato un evento;
 - + `stop(): void`
Metodo che permette di terminare l'ascolto del `Recorder`;

```
+ setConfig(config: RecorderConfig): void
Metodo che permette la modifica della configurazione del Recorder;
Parametri:

* config: RecorderConfig
  È un oggetto che rappresenta la configurazione della registrazione. ;

+ <<Create>> createRecorder(config: RecorderConfig): Recorder
Metodo che permette di creare un oggetto di tipo Recorder a partire da un parametro
di configurazione in formato JSON;
Parametri:

* config: RecorderConfig
  È un oggetto che rappresenta la configurazione della registrazione;
```

- **Relazioni con le altre classi:**

- IN BoolObserver
- OUT RecorderMsg
- OUT RecorderWorker
- OUT RecorderWorkerMsg
- OUT SpeechEndSubject

- **Eventi gestiti:**

- RecorderWorkerMsg
Messaggio mandato da RecorderWorker a Recorder quando l'encoding dell'audio è terminato. Contiene il blob del file audio codificato.

RecorderConfig

- **Nome:** RecorderConfig;
- **Tipo:** Class;
- **Descrizione:** questa classe contiene gli attributi necessari a configurare Recorder;
- **Utilizzo:** fornisce un meccanismo per configurare i parametri di Recorder. ;
- **Attributi:**
 - + threshold: Real
Indica la soglia di volume al di sopra della quale il Recorder deve iniziare la registrazione;
 - + max_silence: Integer
indica il tempo in ms che deve trascorrere prima che il Recorder smetta di registrare a causa di un silenzio (volume al di sotto di threshold). In caso il valore sia -1 (default), il recorder continuerà a registrare per un tempo indefinito;
 - + max_listen: Integer
indica il tempo massimo di registrazione, indicato in millisecondi;

RecorderMsg

- **Nome:** RecorderMsg;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di rappresentare un messaggio che viene mandato dal Recorder al RecorderWorker;

images/ClassRe

- **Utilizzo:** fornisce al **Recorder** un meccanismo per organizzare le informazioni che esso vuole passare a **RecorderWorker**;
- **Attributi:**
 - + **command:** **String**
Attributo contenente il comando da dare al **RecorderWorker**;
 - + **config:** **RecorderWorkerConfig**
Attributo contenente i parametri necessari alla configurazione del **RecorderWorker**. ;
 - + **buffers:** **Float32ArrayArray**
Questo attributo contiene dati grezzi della registrazione audio, da aggiungere ai buffer del Recorder. Contiene un buffer per ogni canale;
 - + **type:** **String**
Attributo contenente il tipo del messaggio;
- **Relazioni con le altre classi:**
 - IN **RecorderWorker**
 - IN **Recorder**

RecorderWorker

- **Nome:** **RecorderWorker**;
- **Tipo:** **Class**;
- **Descrizione:** questa classe è una classe **Worker** che si occupa dell'effettiva registrazione dell'audio;
- **Utilizzo:** fornisce a **Recorder** un thread che si occupa di registrare effettivamente l'audio in modo da non appesantire il thread dell'interfaccia grafica. Eredita la classe **Worker** di JavaScript e ne ridefinisce il metodo **onmessage(Event e)**. (tutti i metodi sono privati perchè vengono usati messaggi per comunicare);
- **Attributi:**
 - **sample_rate:** **int**
Questo attributo contiene la sample rate della registrazione audio nei buffer;
 - **buffers:** **Float32ArrayArray**
Questo attributo contiene i buffer per i dati dell'audio. sono presenti due buffer, uno per ogni canale;
 - **length:** **int**
Questo attributo contiene la lunghezza dei buffer;
- **Metodi:**
 - **encodeWav(samples: Float32Array, mono: boolean): DataView**
Metodo che permette la codifica dell'audio in formato wav a uno o due canali;
Parametri:
 - * **samples:** **Float32Array**
Attributo contenente i dati dei sample audio che verranno utilizzati per la codifica;
 - * **mono:** **boolean**
Attributo contenente un valore booleano che indica se il file audio ha uno o due canali.
In caso il canale sia uno, allora questo attributo dovrà contenere il valore true;

- `clear(): void`

Metodo che permette di eliminare tutti i dati relativi all'audio registrato fino a quel momento;

- `init(config: RecorderWorkerConfig): void`

Metodo che permette di inizializzare la configurazione del `RecorderWorker` in seguito alla ricezione di un messaggio dal `Recorder`;

Parametri:

* `config: RecorderWorkerConfig`

Questo parametro contiene la configurazione;

- `record(): void`

Metodo che permette di iniziare a registrare l'audio dal microfono in seguito ad un messaggio ricevuto dal `Recorder`;

- `getBuffers(): void`

Metodo che permette di ottenere i buffer nei quali sono salvati i dati relativi ai samples dell'audio;

- `exportWav(): void`

Metodo che permette di esportare al `Recorder` il Blob del file wav contenente l'audio della registrazione;

- **Relazioni con le altre classi:**

- IN `Recorder`
- OUT `RecorderWorkerConfig`
- OUT `RecorderWorkerMsg`
- OUT `RecorderMsg`

- **Eventi gestiti:**

- `RecorderMsg`
Messaggio mandato da `Recorder` a `RecorderWorker` per comunicare l'operazione da eseguire in background.

RecorderWorkerConfig

- **Nome:** `RecorderWorkerConfig`;

- **Tipo:** `Class`;

- **Descrizione:** questa classe contiene tutti i parametri relativi alla configurazione di un `RecorderWorker`;

- **Utilizzo:** fornisce al `RecorderWorker` un meccanismo per organizzare le informazioni che esso vuole passare a `Recorder`;

- **Attributi:**

+ `sample_rate: int`

Attributo contenente il valore definito per il sample rate della registrazione in Hz;

- **Relazioni con le altre classi:**

- IN `RecorderWorker`

– IN `RecorderWorkerMsg`

RecorderWorkerMsg

- **Nome:** `RecorderWorkerMsg`;
- **Tipo:** `Class`;
- **Descrizione:** questa classe si occupa di rappresentare un messaggio che viene mandato dal `RecorderWorker` al `Recorder`;
- **Utilizzo:** fornisce al `RecorderWorker` un meccanismo per organizzare le informazioni che esso vuole passare a `Recorder`;
- **Attributi:**
 - + `data: Blob`
Attributo contenente il file audio in formato WAV;
- **Relazioni con le altre classi:**
 - IN `RecorderWorker`
 - IN `Recorder`
 - OUT `RecorderWorkerConfig`

SpeechEndObservable

- **Nome:** `SpeechEndObservable`;
- **Tipo:** `Class`;
- **Descrizione:** classe che rappresenta un `Observable` che notifica gli observer interessati quando l'ospite ha finito di parlare;
- **Utilizzo:** fornisce il meccanismo che permette agli observer di iscriversi per essere notificati quando l'ospite finisce di parlare. Viene creata a partire dal `SpeechEndSubject`, per evitare di dover condividere quest'ultimo;
- **Metodi:**
 - + `subscribe(obs: SpeechEndObserver): Subscription`
Metodo che permette di iscrivere un `Observer` all'`Observable`;
Parametri:
 - * `obs: SpeechEndObserver`
Parametro che rappresenta l'observer che si vuole iscrivere;
- **Relazioni con le altre classi:**
 - IN `LogicObserver`
 - OUT `SpeechEndSubject`

SpeechEndSubject

- **Nome:** `SpeechEndSubject`;
- **Tipo:** `Class`;
- **Descrizione:** questa classe si occupa di notificare l'observer collegato che l'utente ha finito di parlare;

images/ClassSp

- **Utilizzo:** fornisce un meccanismo che permette l'invio del file audio contenente il messaggio dell'utente una volta che quest'ultimo ha terminato di parlare. L'observer collegato è LogicObserver;
- **Padre:** Subject;
- **Metodi:**
 - + next(val: boolean): void
Questo metodo permette di notificare il LogicObserver;
Parametri:
 - * val: boolean
Parametro contenente il valore con il quale notificare il LogicObserver;
 - + subscribe(obs: BoolObserver): Subscription
Questo metodo permette di iscrivere il LogicoObserver;
Parametri:
 - * obs: BoolObserver
Parametro contenente il valore con il quale il LogicObserver dev'essere iscritto;
- **Relazioni con le altre classi:**
 - IN Recorder
 - IN SpeechEndObservable

Client::Registry

Package contenente i registri che vanno a memorizzare le applicazioni.

Classi

ApplicationLocalRegistry

- **Nome:** ApplicationLocalRegistry;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di mantenere una lista degli ApplicationPackage disponibili. ;
- **Utilizzo:** fornisce a LocalApplicationRegistryClient i metodi necessari ad inserire, rimuovere ed ottenere degli ApplicationPackage. Implementa un Registry all'interno del client, semplificando il compito del RegistryClient ed eliminando la necessità di effettuare richieste HTTP per recuperare un ApplicationPackage. ;
- **Attributi:**
 - pkgs: ApplicationPackageAssocArray
Attributo contenente l'Array dei nomi delle applicazioni contenute nel registry;
- **Metodi:**
 - + <<Create>> createLocalRegistry(): ApplicationLocalRegistry
Metodo che permette di istanziare un ApplicationLocalRegistry;
 - + query(name: String): ApplicationPackage
Metodo che permette di interrogare il LocalRegistry a partire dal nome dell'applicazione, ottenendo l' ApplicationPackage relativo ad essa;
Parametri:

```

* name: String
    Parametro contenente il nome dell'applicazione da recuperare;
+ register(name: String, pkg: ApplicationPackage): void
Metodo che permette la registrazione di una applicazione nel LocalRegistry;
Parametri:

* name: String
    Parametro contenente il nome dell'applicazione da registrare;

* pkg: ApplicationPackage
    Parametro contenente il package relativo all'applicazione da registrare;
+ remove(name: String): void
Metodo che permette di rimuovere un'applicazione a partire dal suo nome;
Parametri:

* name: String
    Parametro contenente il nome dell'applicazione da rimuovere;

```

Client::TTS

Package contenente le componenti che realizzano il text to speech.

Classi

Player

- **Nome:** Player;
- **Tipo:** Class;
- **Descrizione:** questa classe si occupa di riprodurre la risposta, fornita dal sistema, all'ospite;
- **Utilizzo:** fornisce al client funzionalità di riproduzione del file audio di risposta, da parte del sistema, relativo a ciò che l'ospite comunica. Permette all'ospite di fermare e riavviare la riproduzione audio e di impostare il volume. ;
- **Attributi:**
 - **tts:** SpeechSynthesis
Attributo contenente uno SpeechSynthesis di JavaScript Web Speech API il quale fornisce dei metodi per la gestione del file audio. Per la relativa documentazione, consultare questa pagina <https://developer.mozilla.org/en-US/docs/Web/API/SpeechSynthesis>;
 - **subject:** BoolSubject
Attributo contenente un BoolSubject, il quale emette item del tipo:
 - * TRUE quando l'utente sta parlando;
 - * FALSE altrimenti.

Questo attributo serve per disattivare il Recorder durante la riproduzione del file audio relativo alla risposta, fornita dal sistema;
 - **options:** TTSTConfig
Attributo contenente l'insieme delle opzioni relative alla configurazione del Player;
- **Metodi:**
 - + **pause():** void
Metodo che permette di mettere in pausa la riproduzione dell'audio;

```
+ <<Create>> createPlayer(conf: TTSTConfig, speech_synthesis: SpeechSynthesis):
  Player
```

Metodo che si occupa di creare un oggetto di tipo `Player` con una determinata configurazione. Permette di fornire un'oggetto `SpeechSynthesis` al `Player`, effettuandone quindi la dependency injection a costruttore;

Parametri:

```
* conf: TTSTConfig
```

Rappresenta l'audio da utilizzare nella creazione di un `Player`;

```
* speech_synthesis: SpeechSynthesis
```

Parametro che rappresenta l'oggetto `SpeechSynthesis` di cui viene effettuata la constructor-based dependency injection;

```
+ isPlaying(): boolean
```

Metodo che permette di capire se il `Player` sta riproducendo un audio o meno;

```
+ speak(text: String): void
```

Metodo che fa pronunciare una frase dal `Player`;

Parametri:

```
* text: String
```

Questo parametro contiene il testo da pronunciare;

```
+ cancel(): void
```

Metodo che permette di bloccare la riproduzione dell'audio;

```
+ resume(): void
```

Metodo che permette di riprendere la riproduzione audio dal punto in cui era stata interrotta;

```
+ setConfig(conf: TTSTConfig): void
```

Metodo che permette di modificare la configurazione;

Parametri:

```
* conf: TTSTConfig
```

È l'insieme delle opzioni da passare;

```
+ getVoices(): SpeechSynthesisVoiceArray
```

Metodo che restituisce una lista con le voci disponibili nel sistema per la sintesi vocale;

```
+ getObservable(): BoolObservable
```

Metodo che ritorna l'`Observable` relativo all'attributo `subject`;

- **Relazioni con le altre classi:**

- IN `PlayerObserver`
- OUT `TTSTConfig`
- OUT `BoolObservable`
- OUT `BoolSubject`

PlayerObserver

- **Nome:** `PlayerObserver`;
- **Tipo:** `Class`;

- **Descrizione:** questa classe si occupa di inviare l'audio contenente la risposta fornita dal sistema al Player. È la classe che funziona da Observer;
- **Utilizzo:** fornisce un meccanismo per l'invio dell'audio relativo alla risposta fornita dal sistema. ;
- **Padre:** ObserverAdapter;
- **Metodi:**
 - + next(callback: function(val: ResponseBody): void): void
Questo metodo, all'arrivo della notifica dal DataArrivedSubject, permette di passare al PlayerObserver un'operazione da eseguire. ;
 - Parametri:
 - * callback: function(val: ResponseBody): void
Parametro contenente la funzione di callback da eseguire all'arrivo dei dati;
- **Relazioni con le altre classi:**
 - OUT Player
 - OUT DataArrivedObservable

TTSTConfig

- **Nome:** TTSTConfig;
- **Tipo:** Class;
- **Descrizione:** questa classe contiene tutti i parametri relativi alla configurazione di un Player;
- **Utilizzo:** fornisce al Player un meccanismo per gestire i parametri relativi alla configurazione per la riproduzione di un file audio. Per la relativa documentazione, consultare la pagina <https://developer.mozilla.org/it/docs/Web/API/SpeechSynthesisUtterance>;
- **Attributi:**
 - + lang: String
Attributo contenente la lingua dell'audio che deve essere pronunciato dal Player. ;
 - + pitch: float
Attributo contenente il valore del pitch (intonazione) del file da riprodurre;
 - + rate: int
Attributo contenente la velocità alla quale il file audio deve essere pronunciato. ;
 - + voice: SpeechSynthesisVoice
Attributo contenente la voce con la quale il file audio deve essere pronunciato. ;
 - + volume: int
Attributo contenente il volume al quale il file audio deve essere pronunciato;
- **Relazioni con le altre classi:**
 - IN Player

Client::Utility

Package contenente classi e interfacce, dallo scopo generico, utili ad altri package del client.

Classi

BoolObservable

- **Nome:** BoolObservable;
- **Tipo:** Class;
- **Descrizione:** ereditata da Observable, con eventi di tipo boolean;
- **Utilizzo:** viene utilizzata come observable;
- **Metodi:**
 - + subscribe(obs: BoolObserver): Subscription
Iscrive l'Observer all'Observable;
Parametri:
 - * obs: BoolObserver
OsservatoreBool;
- **Relazioni con le altre classi:**
 - IN Player
 - IN BoolObserver
 - OUT BoolSubject

BoolObserver

- **Nome:** BoolObserver;
- **Tipo:** Class;
- **Descrizione:** questa classe;
- **Utilizzo:** fornisce un meccanismo per eseguire una funzione all'arrivo di una notifica dal BoolObservable;
- **Metodi:**
 - + next(function(val: boolean)): void: void
Questo metodo, all'arrivo della notifica dal BoolObservable, permette di passare al BoolObserver un'operazione da eseguire;
Parametri:
 - * function(val: boolean): void
Parametro contenente la funzione da eseguire. ;
- **Relazioni con le altre classi:**
 - OUT Recorder
 - OUT BoolObservable

BoolSubject

- **Nome:** BoolSubject;
- **Tipo:** Class;
- **Descrizione:** classe che eredita da Subject, che fa uso di valori boolean;
- **Utilizzo:** viene utilizzata nella stessa maniera di Subject, con valori di tipo boolean;

images/ClassB

- **Metodi:**

- + `next(val: boolean): void`

- Questo metodo permette di notificare il BoolSubject;

- Parametri:

- * `val: boolean`

- Parametro contenente il valore con il quale notificare il BoolSubject;

- + `subscribe(obs: BoolObserver): Subscription`

- Questo metodo permette di iscrivere il BoolObserver;

- Parametri:

- * `obs: BoolObserver`

- Parametro contenente il valore con il quale iscrivere il BoolObserver;

- **Relazioni con le altre classi:**

- IN `Player`

- IN `BoolObservable`

images/ClassOb

RxJS

Libreria esterna

Classi

Observable

- **Nome:** Observable;
- **Tipo:** Class;
- **Descrizione:** observable di RxJs;
- **Utilizzo:** ;
- **Figli:** UserObservable, ConversationObservable, GuestObservable, TaskObservable, RuleObservable, AgentObservable, ErrorObservable;

Observer

- **Nome:** Observer;
- **Tipo:** Class;
- **Descrizione:** observer di RxJS;
- **Utilizzo:** ;

Subject

- **Nome:** Subject;
- **Tipo:** Class;
- **Descrizione:** rxJS5;
- **Utilizzo:** ;

- **Figli:** `SpeechEndSubject`, `DataArrivedSubject`;

images/ClassSp

WatsonDeveloperCloud

SDK node.js per l'accesso ai servizi cloud IBM Watson.

Classi

SpeechToTextV1

- **Nome:** `SpeechToTextV1`;
- **Tipo:** `Class`;
- **Descrizione:** classe che rappresenta il modulo node.js che permette l'utilizzo del servizio di STT di IBM;
- **Utilizzo:** viene utilizzata per accedere al servizio STT fornito da IBM. Permette di interrogare tale servizio utilizzando dei callback oppure utilizzando gli streams di node.js. Per ulteriori informazioni, consultare la seguente pagina: <https://github.com/watson-developer-cloud/node-sdk#speech-to-text>;

DA MODIFICARE DA MODIFICARE DA MODIFICARE DA MODIFICARE.png DA
MODIFICARE DA MODIFICARE DA MODIFICARE DA MODI-
FICARE.png DA MODIFICARE DA MODIFICARE DA MODIFICARE DA MODIFICARE.png



images/ClassVArequestAPIBody DA MODIFICARE DA MODIFICARE DA MODIFICARE DA MODIFICARE.png

Figura 30: Back-end::APIGateway::VArequestAPIBody DA MODIFICARE DA MODIFICARE
DA MODIFICARE DA MODIFICARE

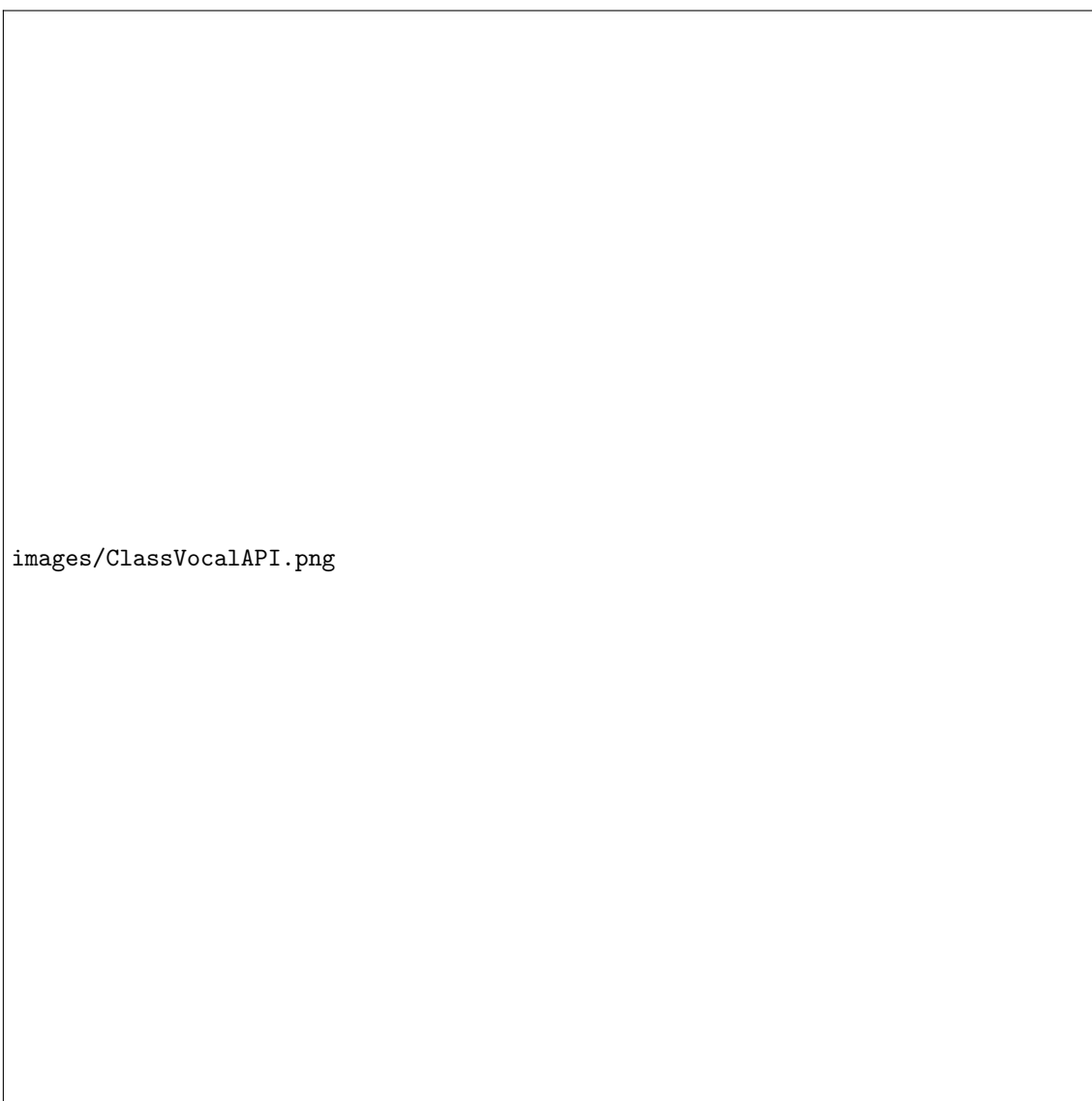
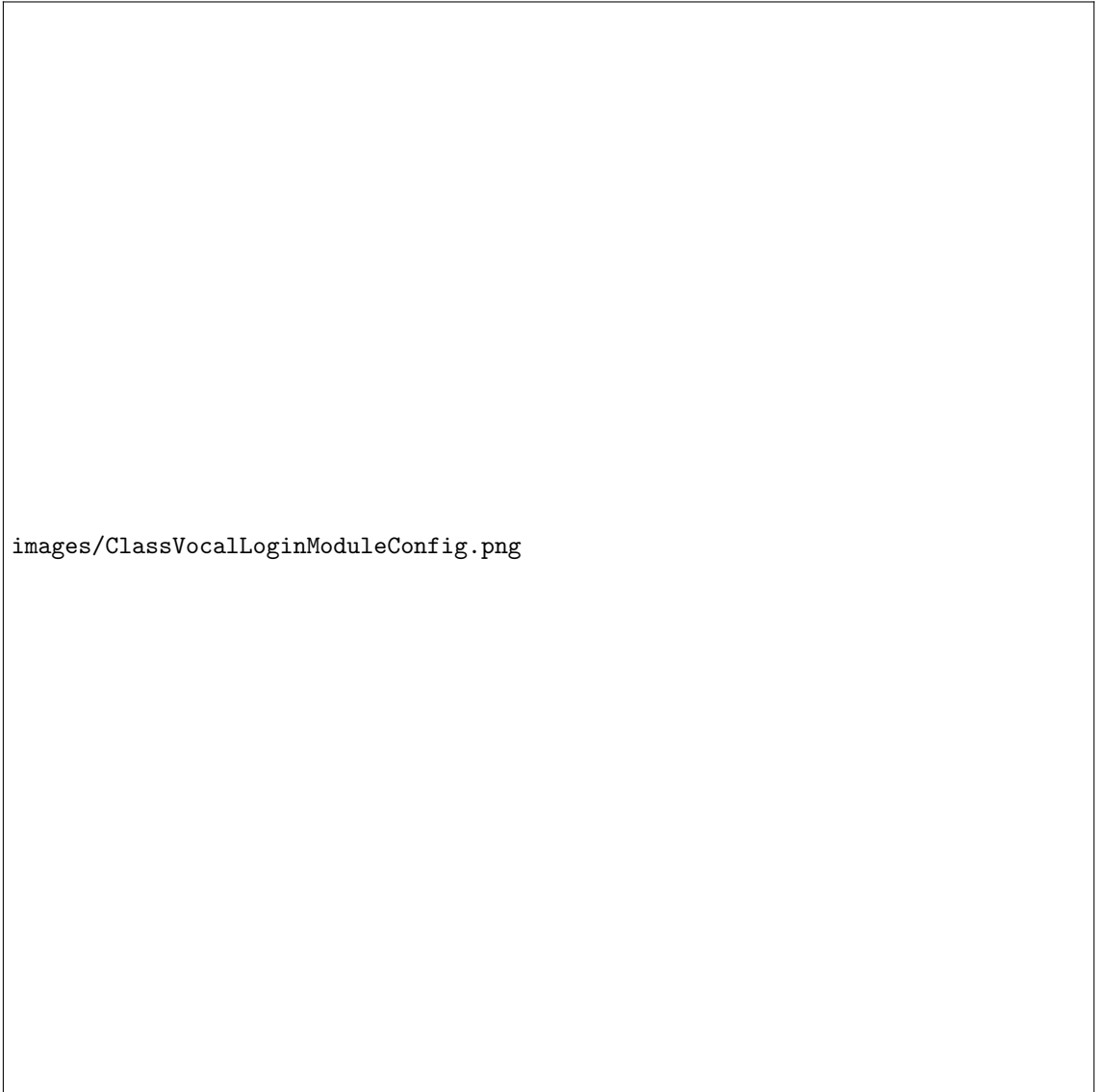
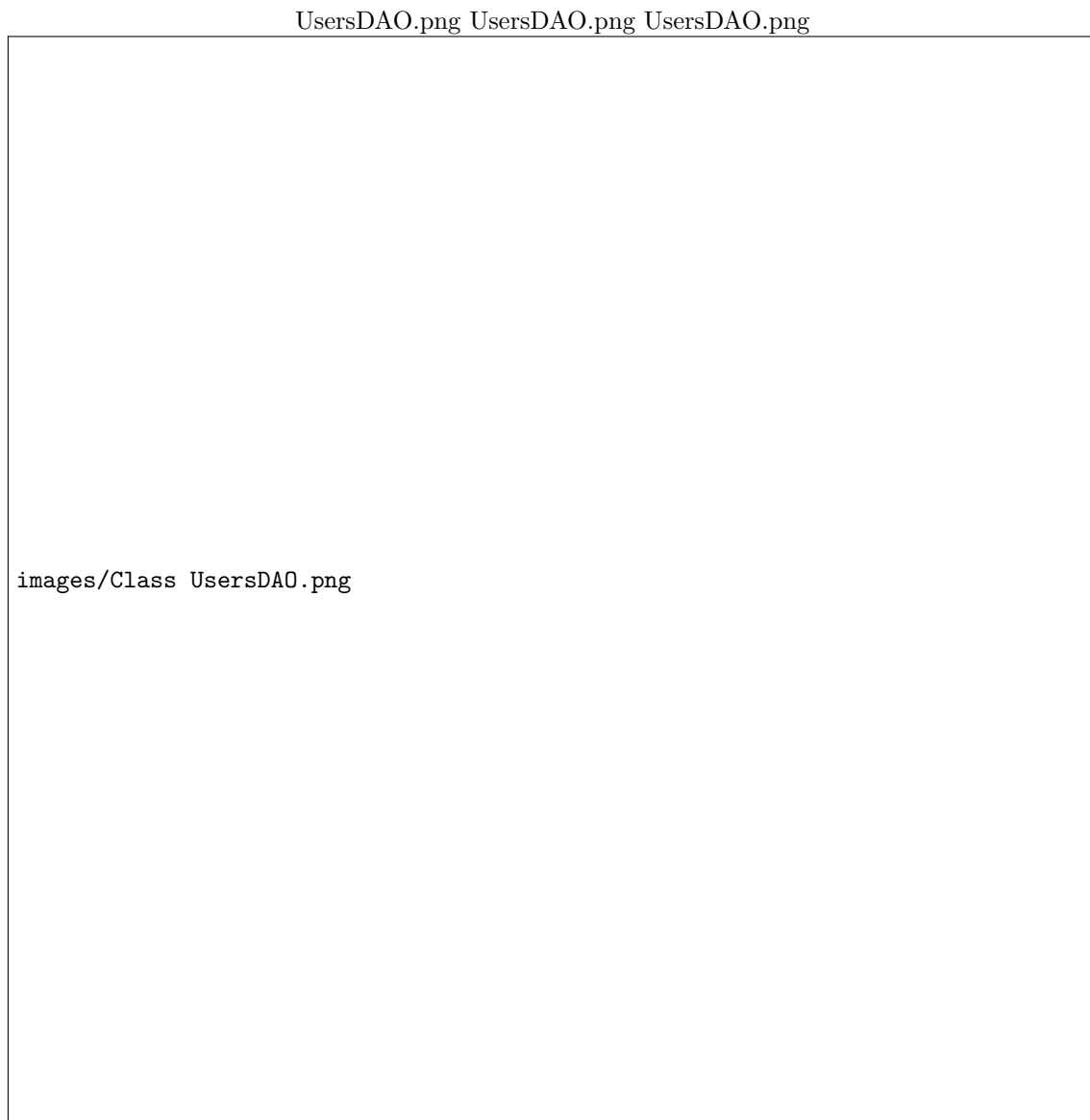


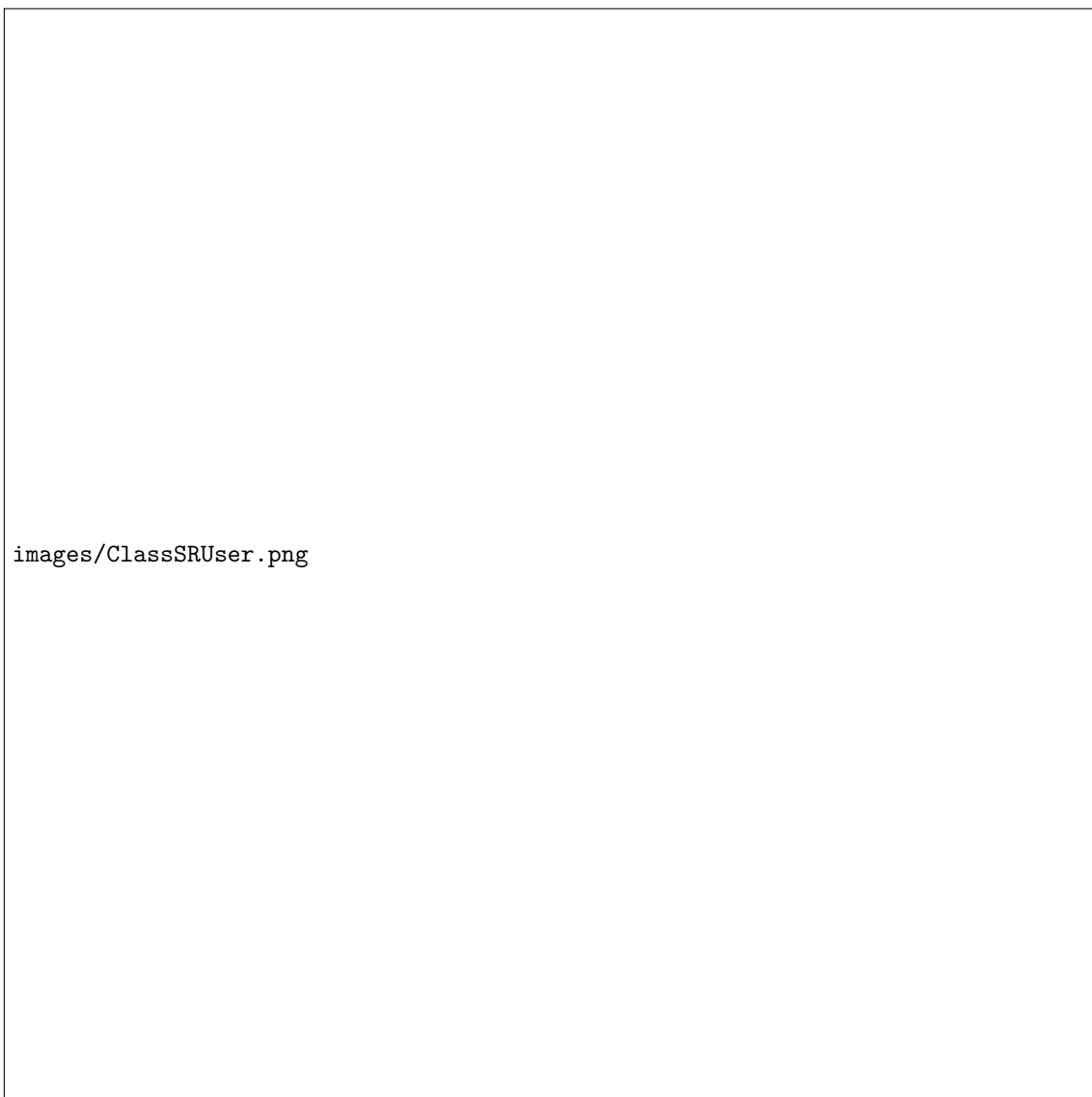
Figura 31: Back-end::APIGateway::VocalAPI



images/ClassVocalLoginModuleConfig.png

Figura 32: Back-end::APIGateway::VocalLoginModuleConfig

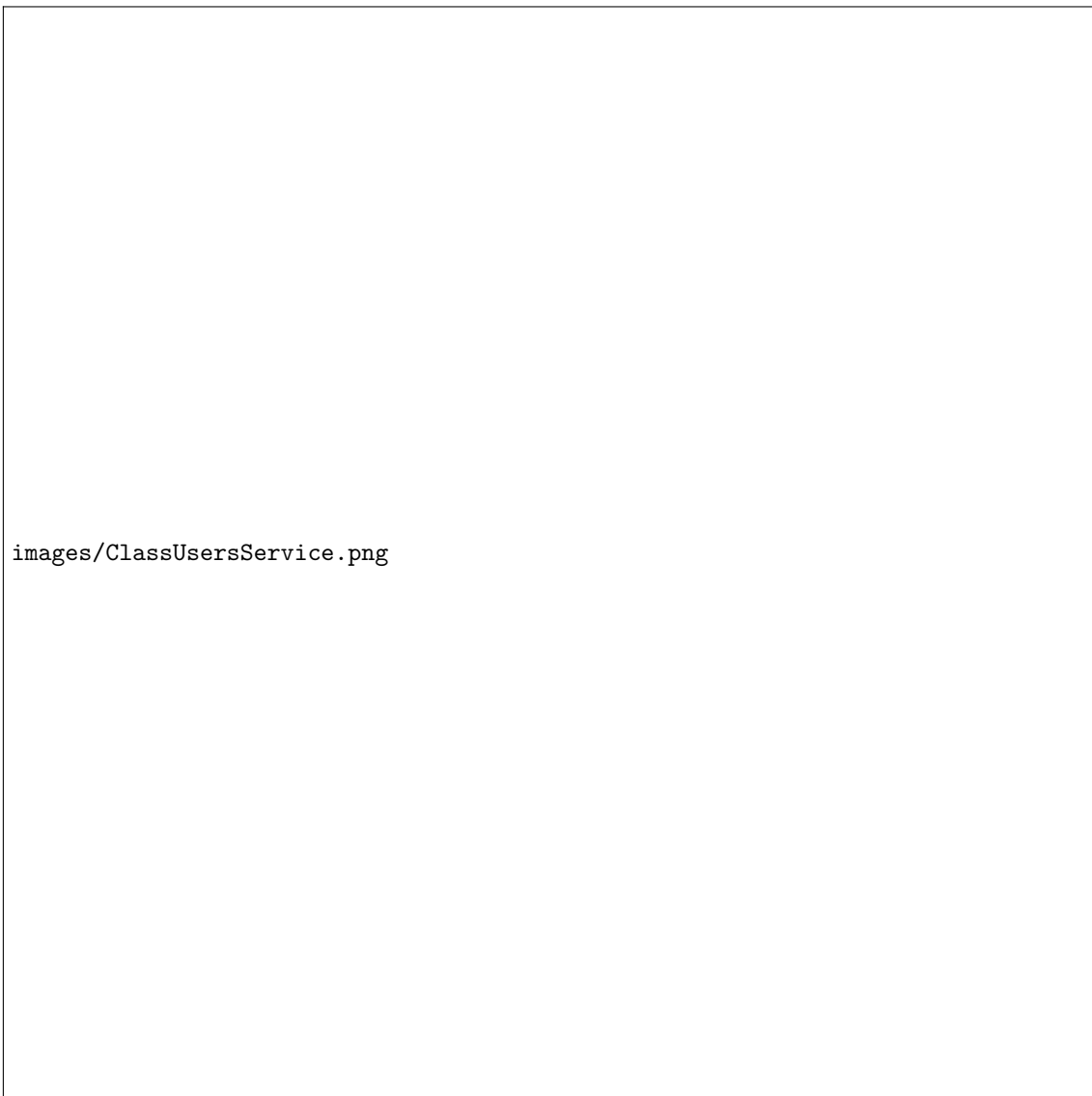
**Figura 33:** Back-end::Auth:: UsersDAO

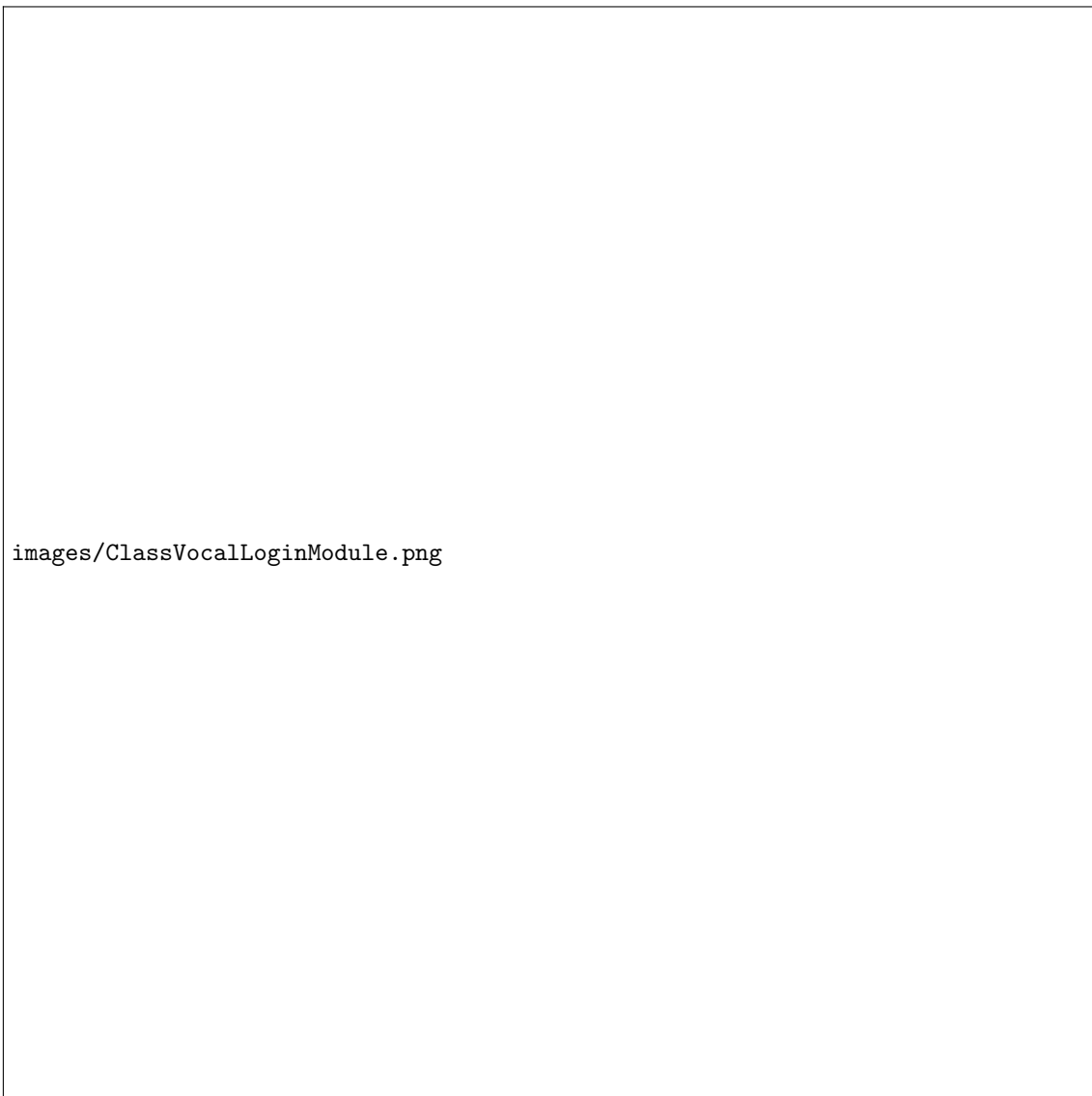
**Figura 34:** Back-end::Auth::SRUser

**Figura 35:** Back-end::Auth::User



Figura 36: Back-end::Auth::UsersDAODynamoDB

**Figura 37:** Back-end::Auth::UserService

**Figura 38:** Back-end::Auth::VocalLoginModule

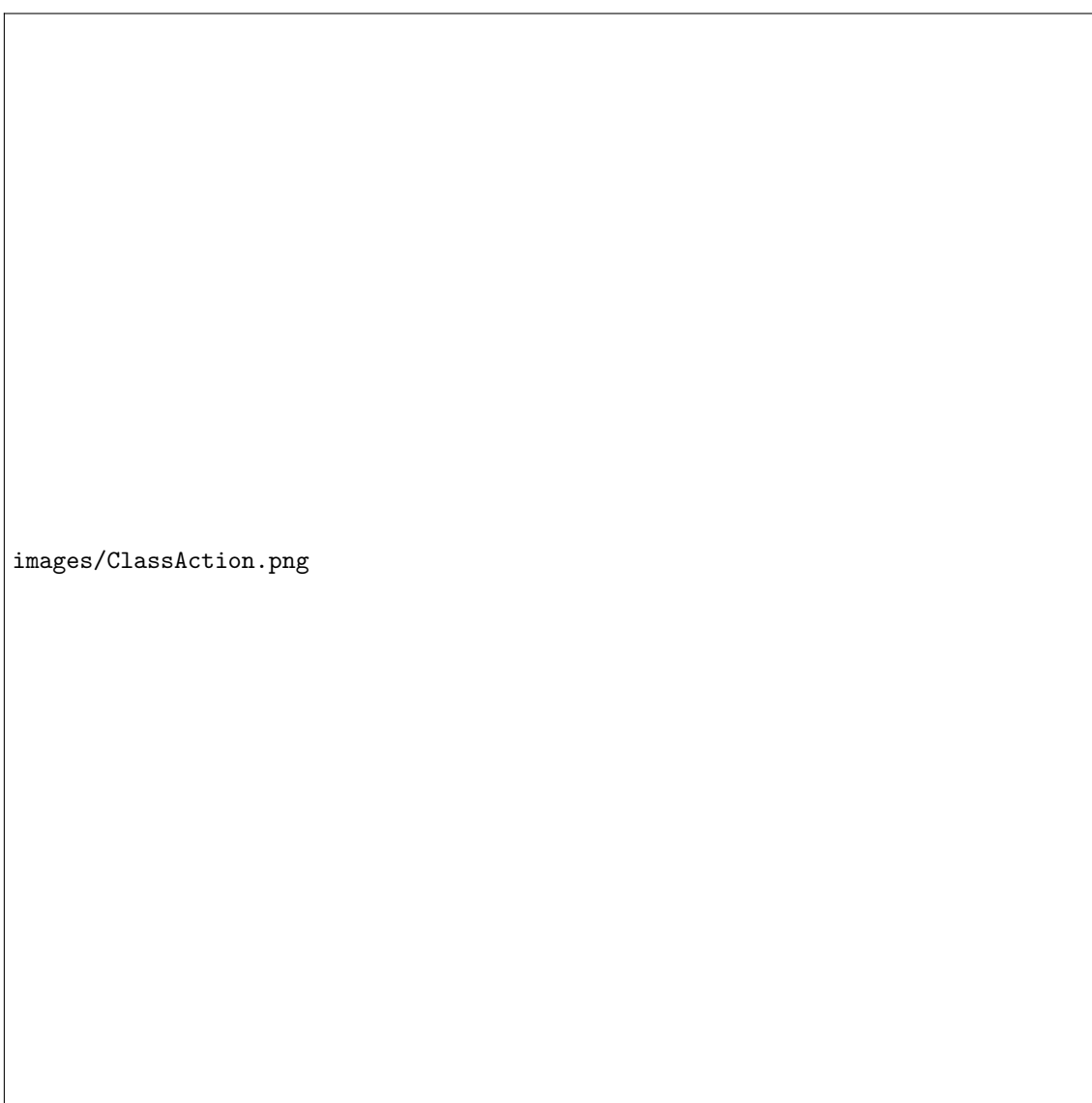
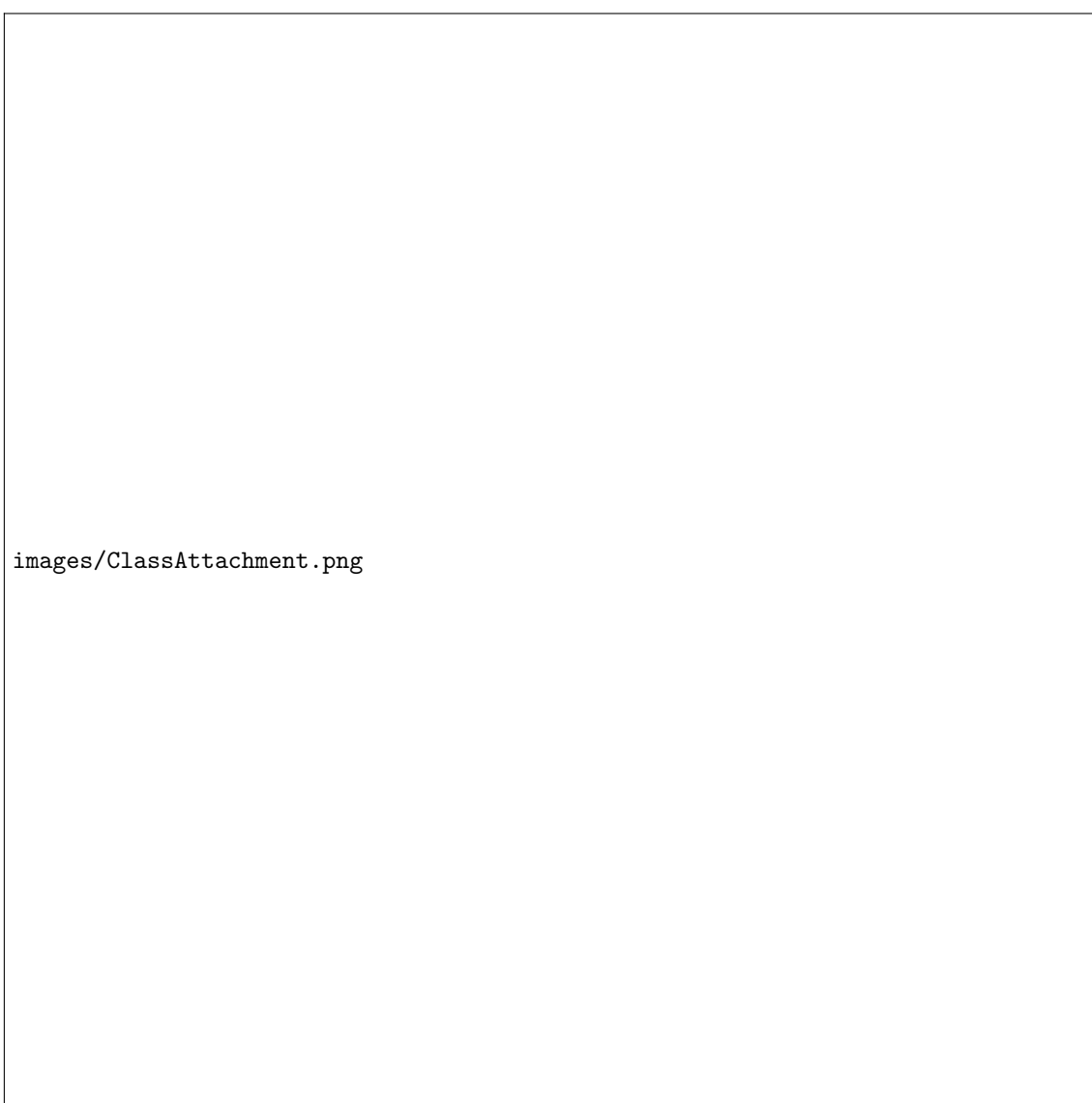
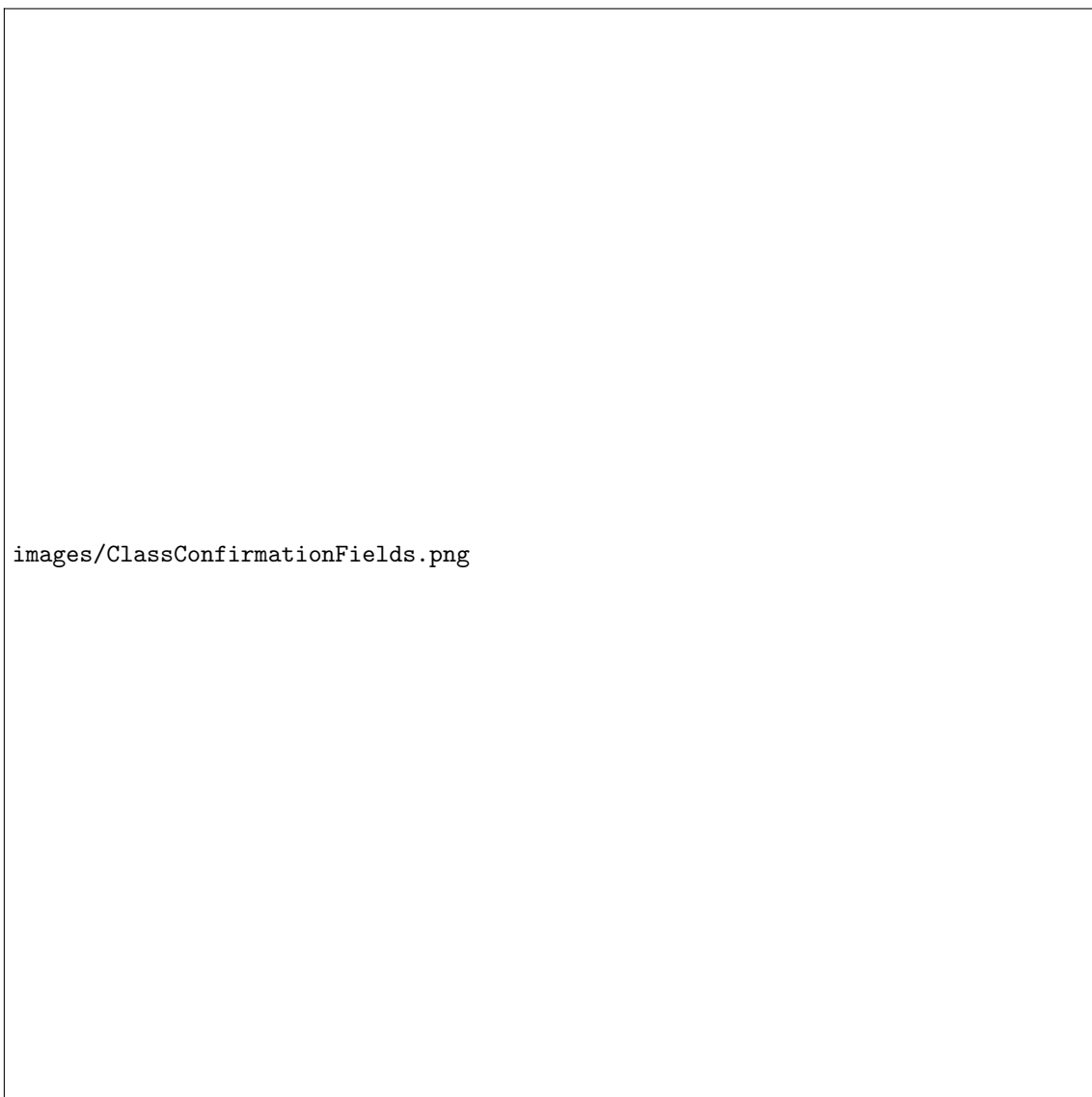


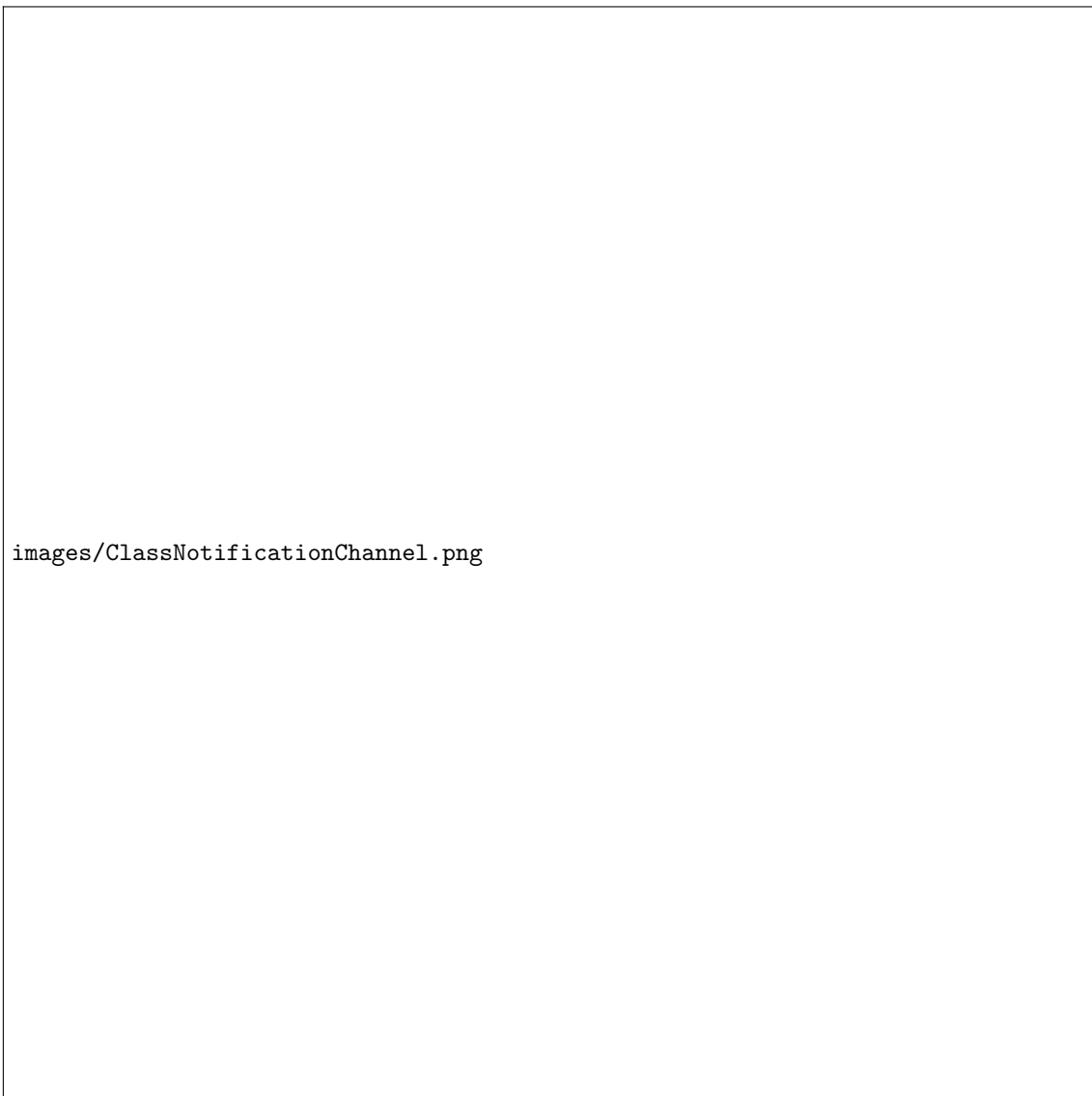
Figura 39: Back-end::Notifications::Action

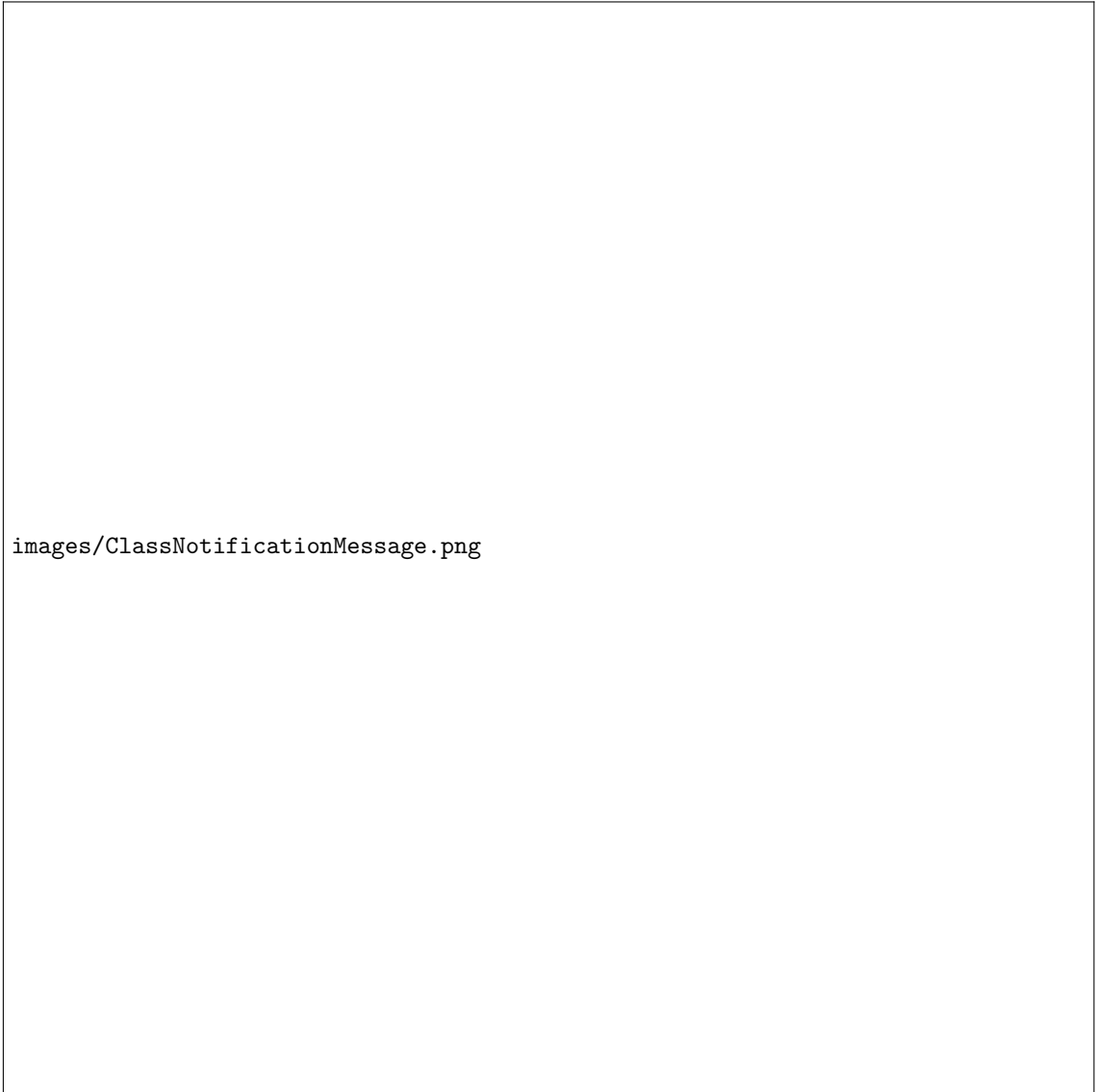


images/ClassAttachment.png

Figura 40: Back-end::Notifications::Attachment

**Figura 41:** Back-end::Notifications::ConfirmationFields

**Figura 42:** Back-end::Notifications::NotificationChannel



images/ClassNotificationMessage.png

Figura 43: Back-end::Notifications::NotificationMessage

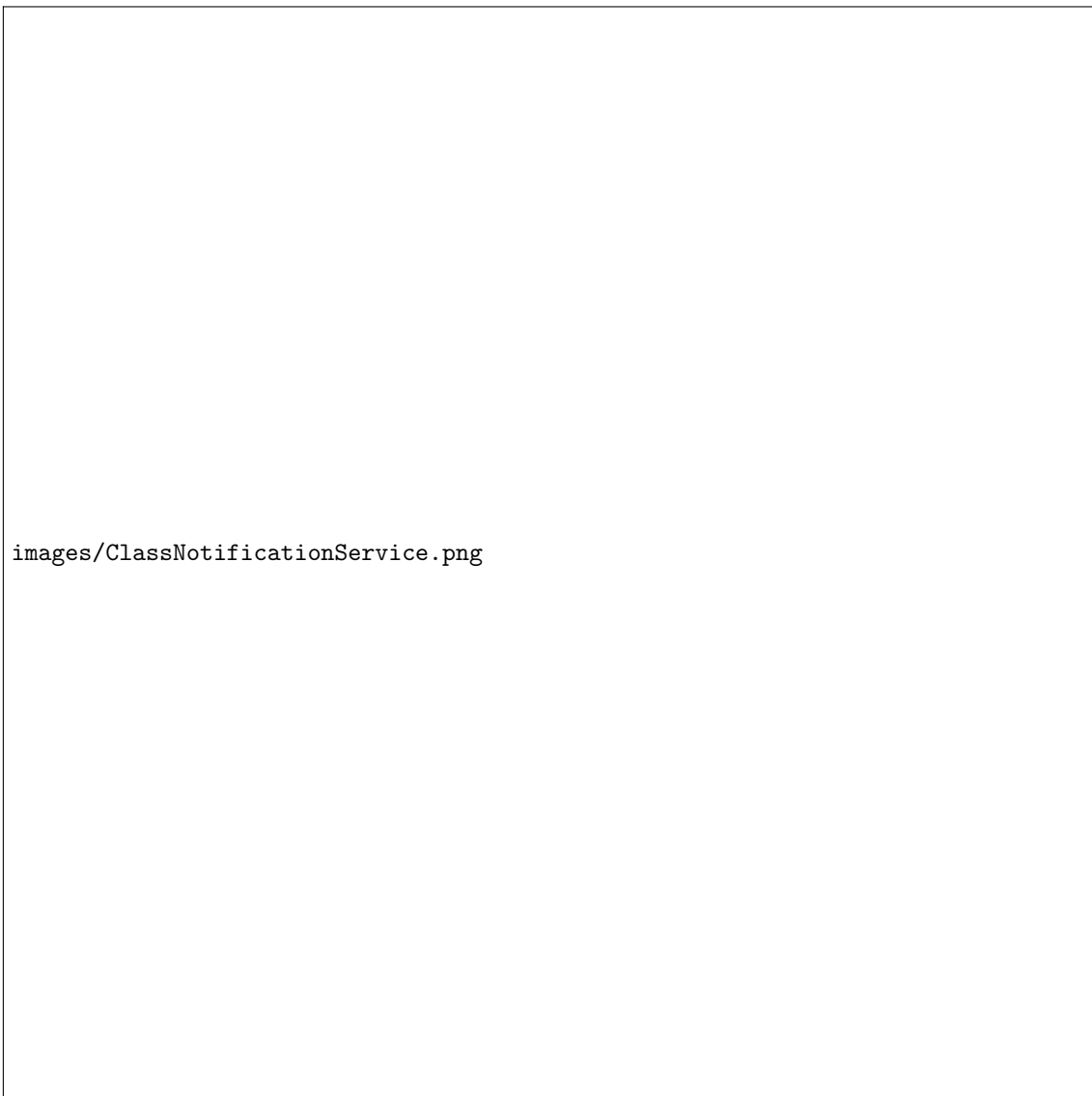


Figura 44: Back-end::Notifications::NotificationService

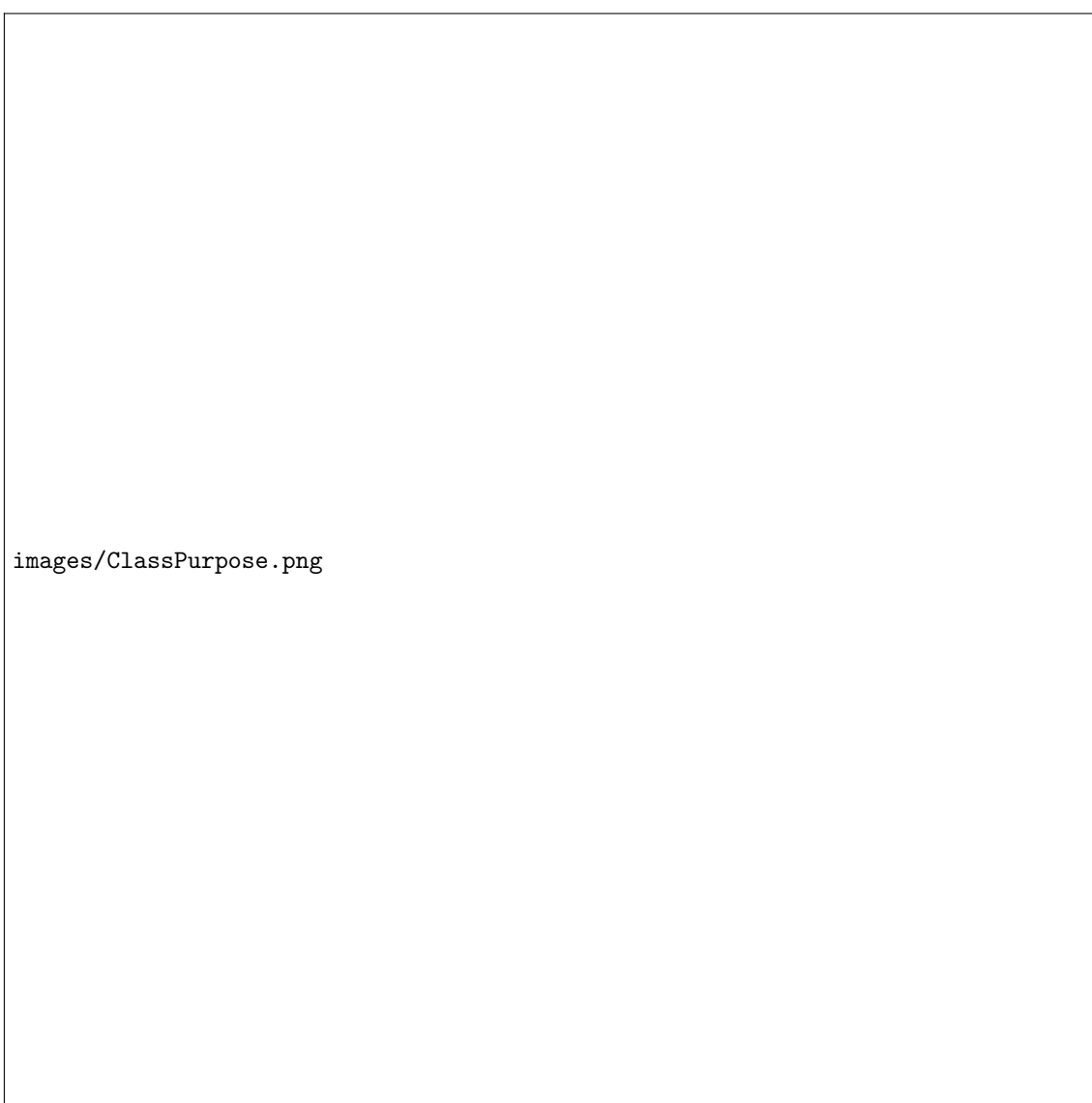
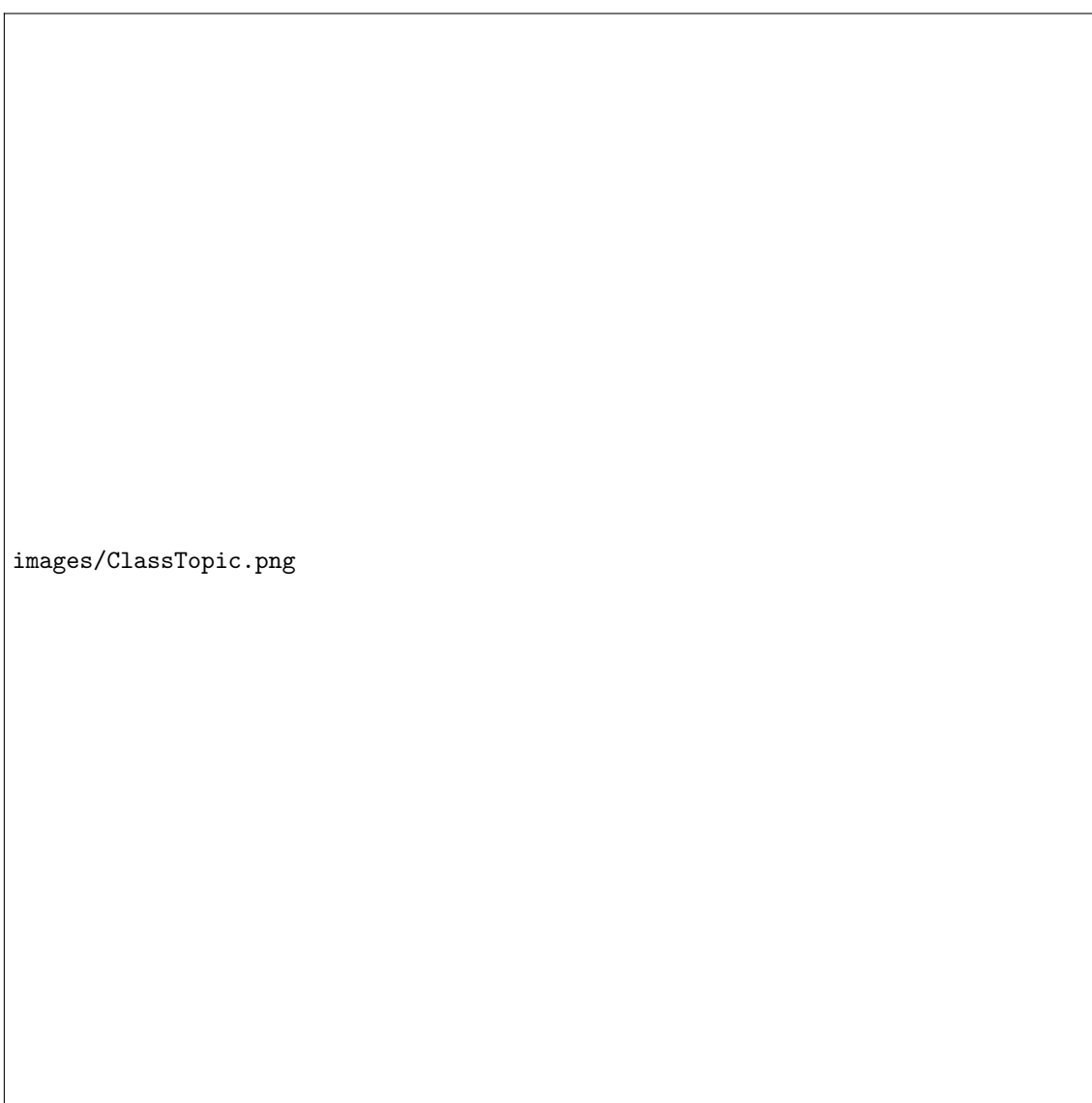
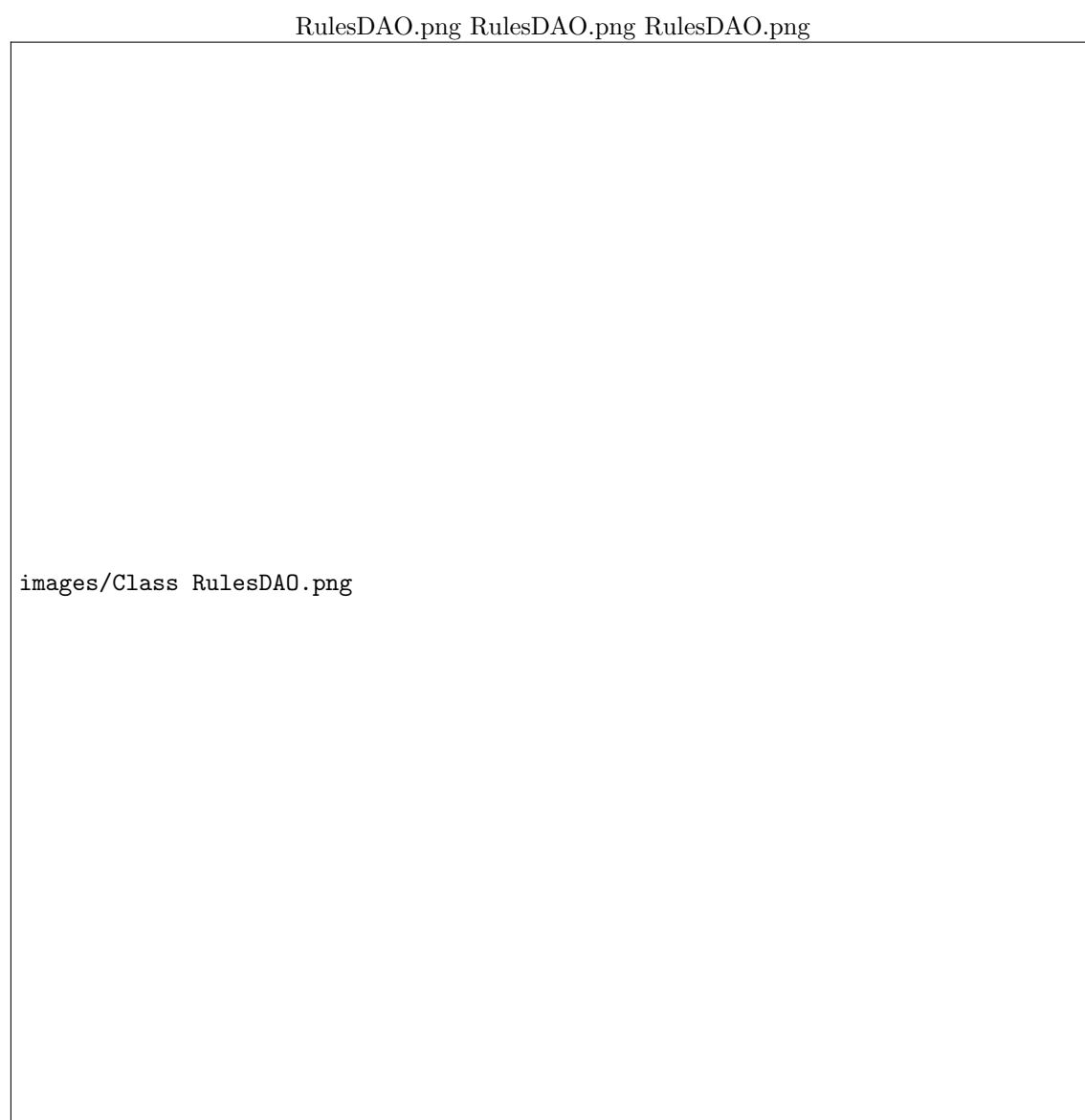
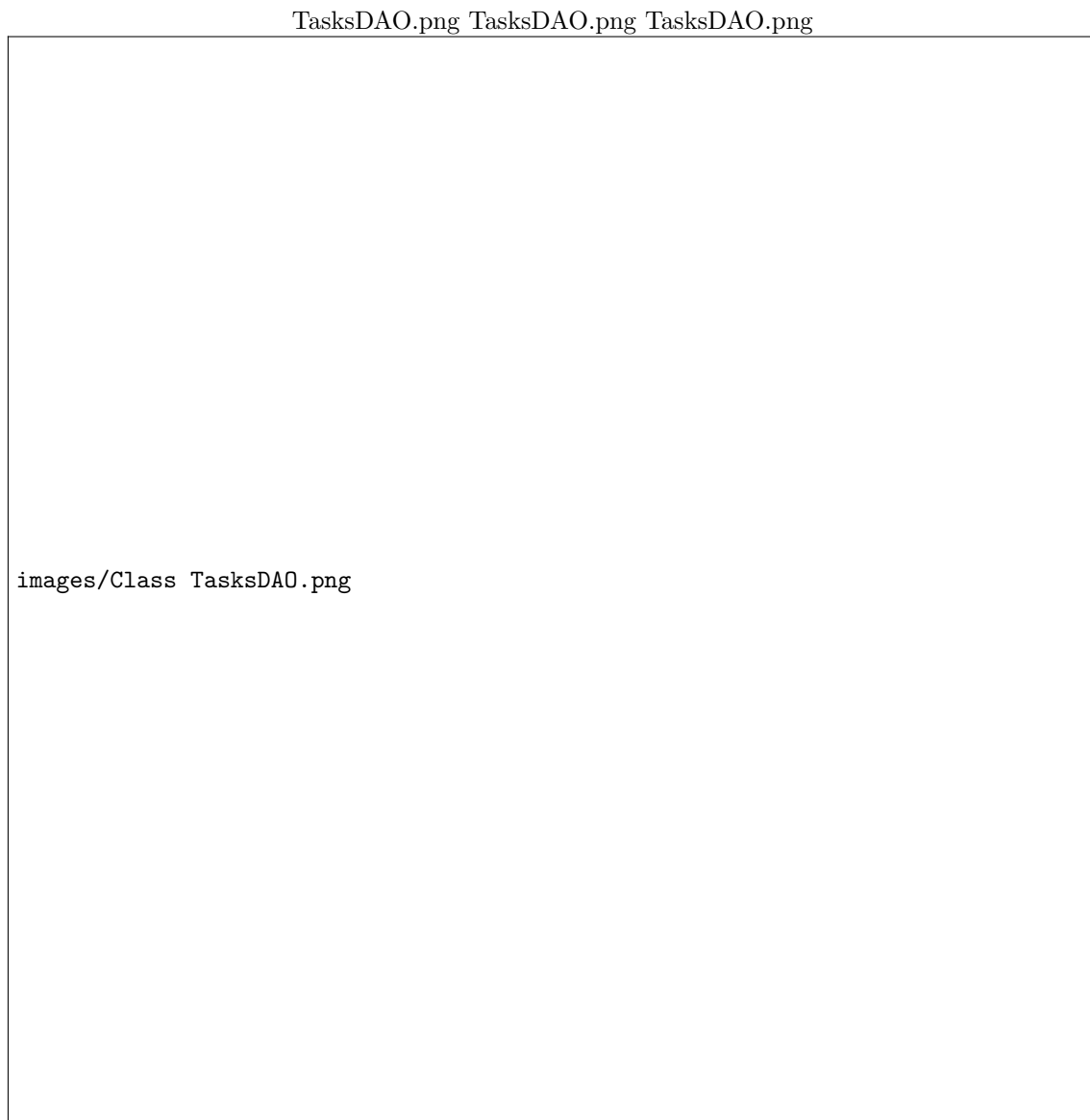


Figura 45: Back-end::Notifications::Purpose

**Figura 46:** Back-end::Notifications::Topic

**Figura 47:** Back-end::Rules:: RulesDAO

**Figura 48:** Back-end::Rules:: TasksDAO

**Figura 49:** Back-end::Rules::Rule

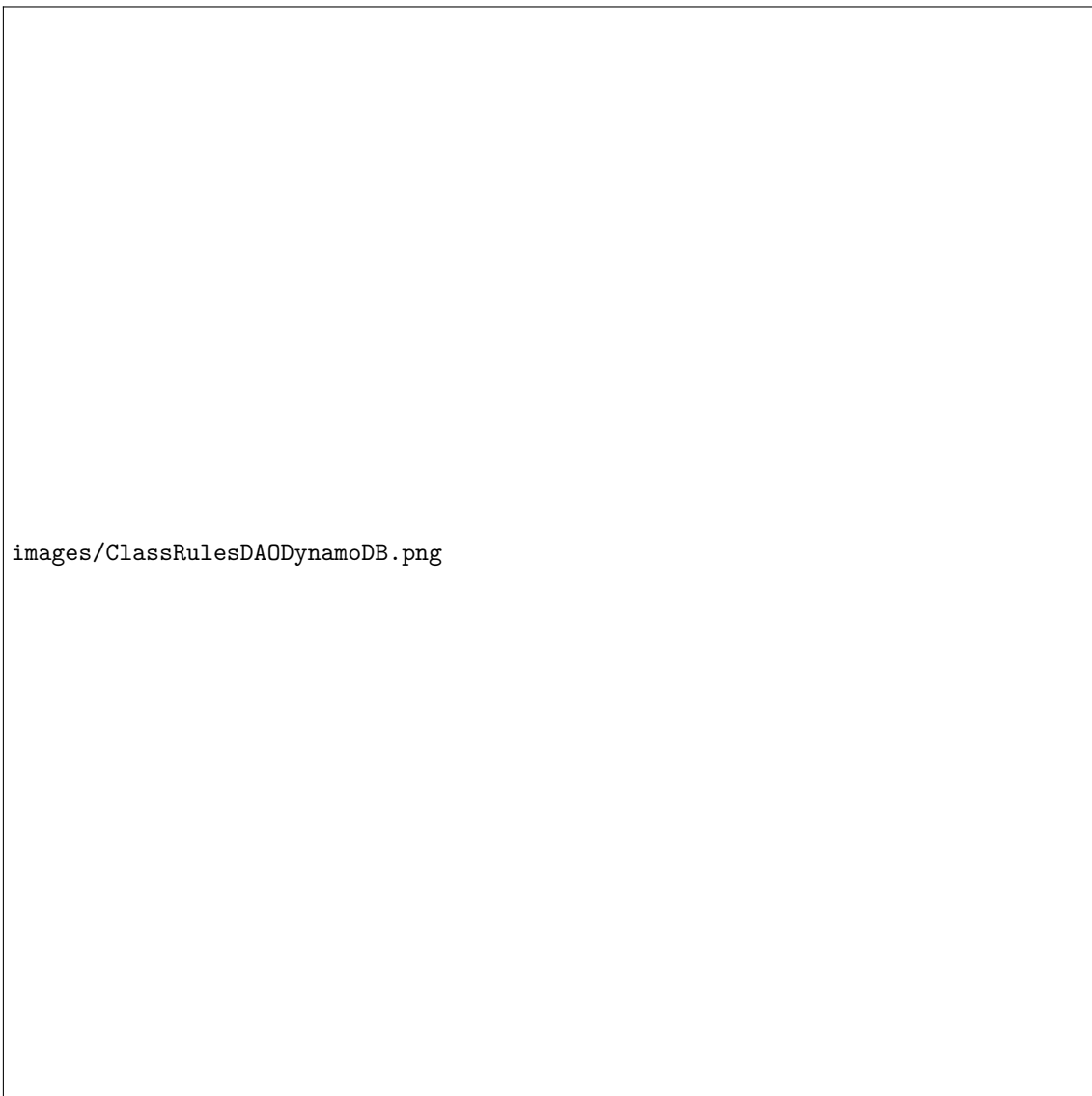
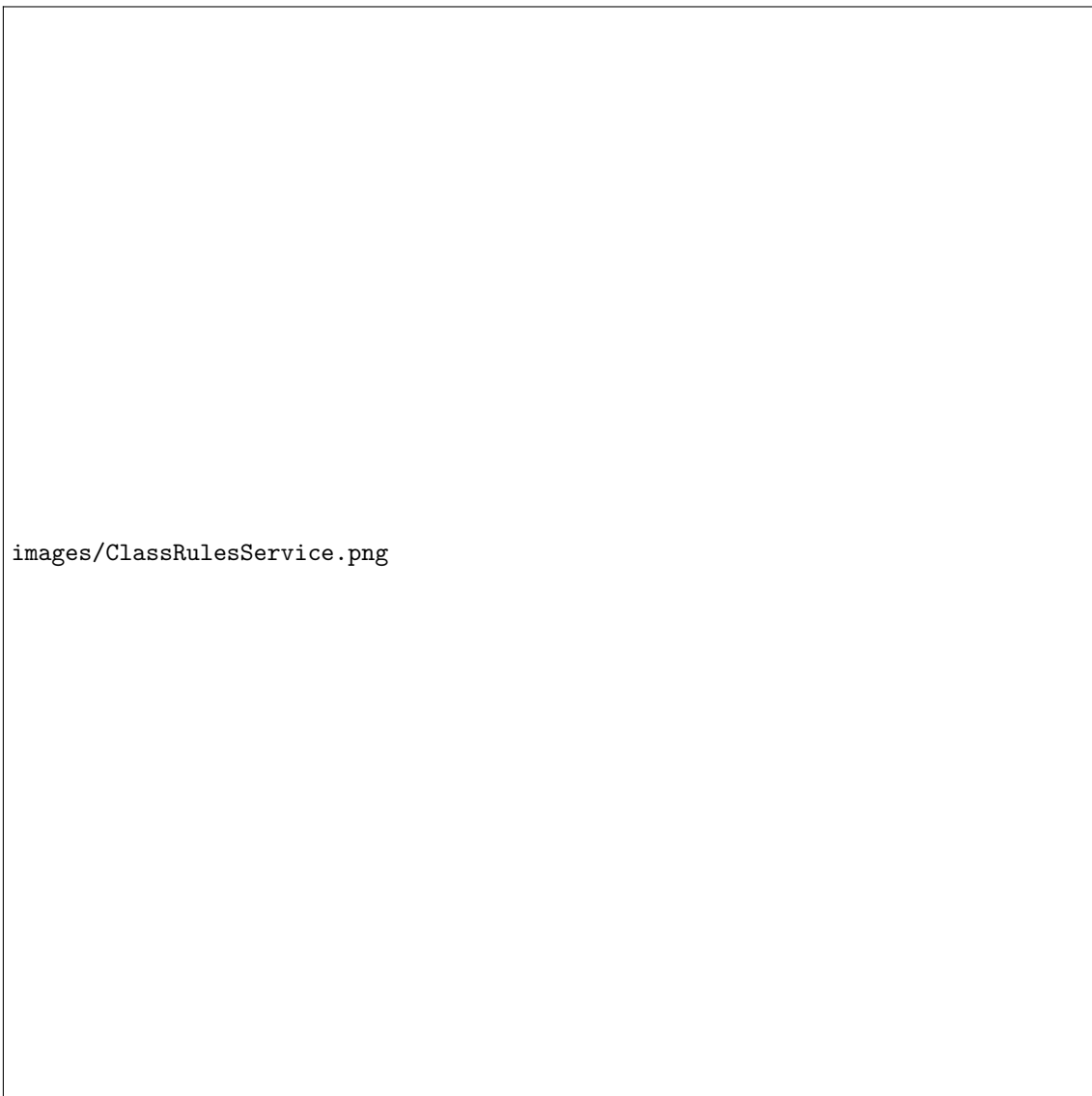
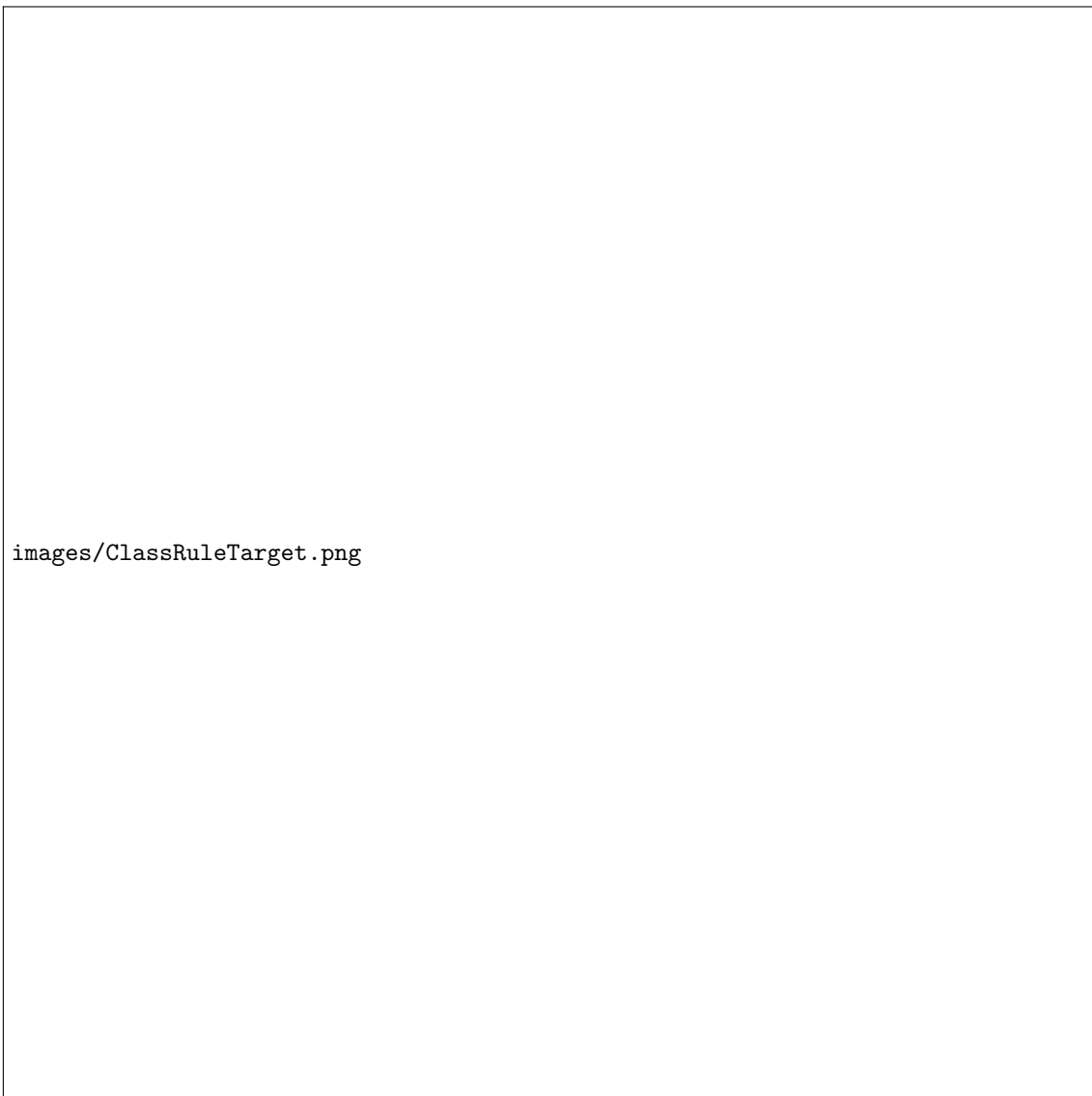
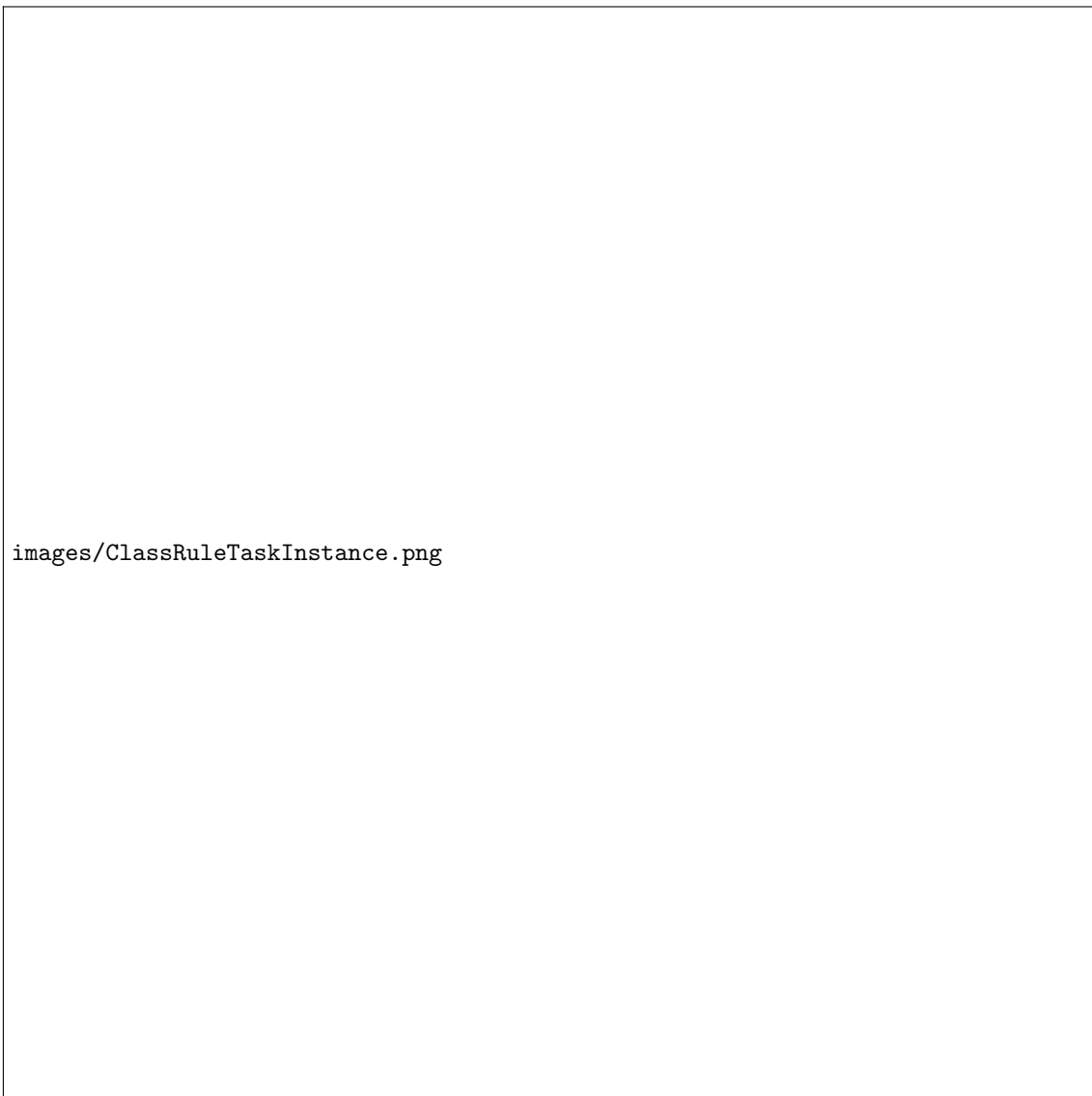
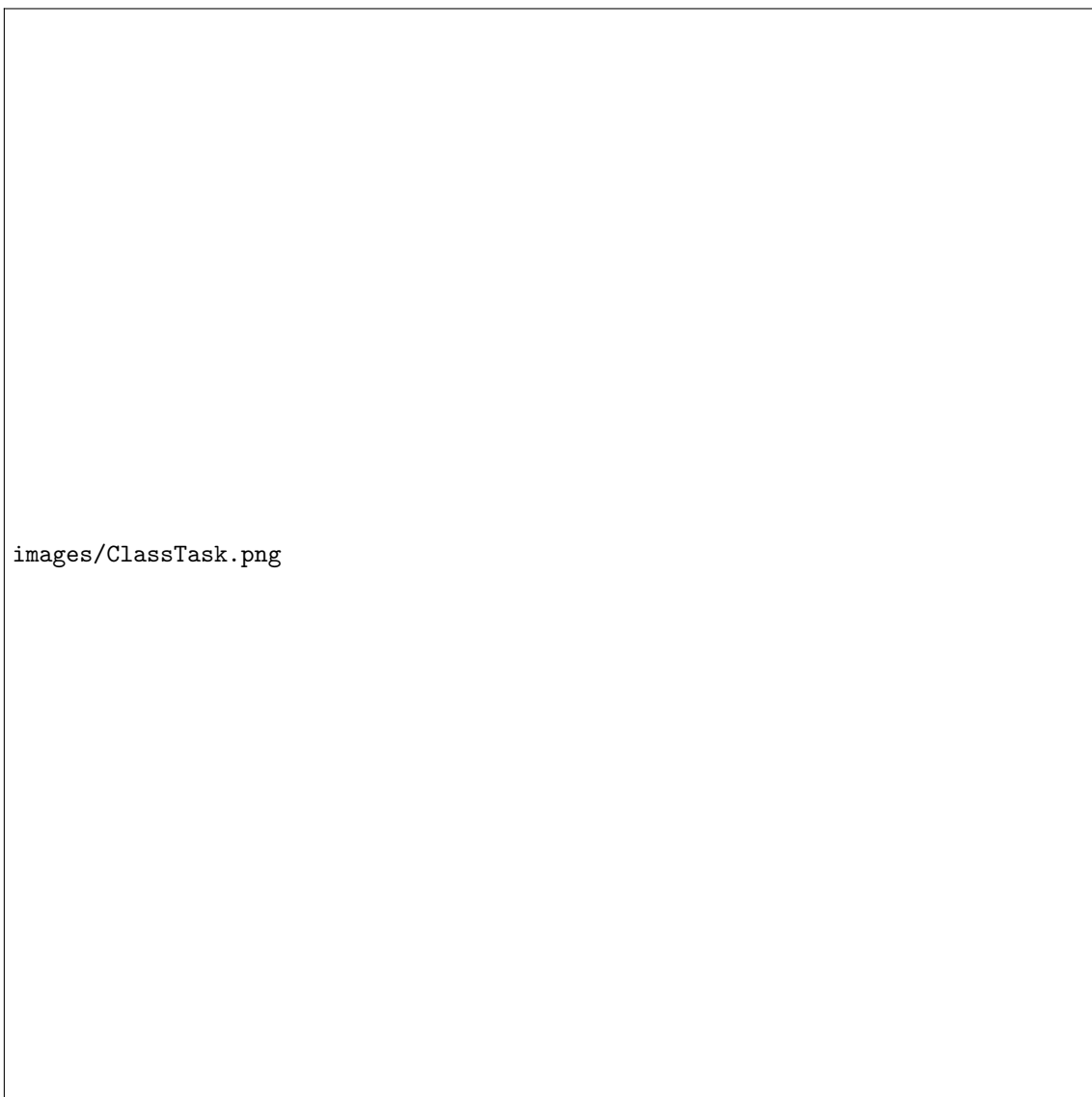


Figura 50: Back-end::Rules::RulesDAODynamoDB

**Figura 51:** Back-end::Rules::RulesService

**Figura 52:** Back-end::Rules::RuleTarget

**Figura 53:** Back-end::Rules::RuleTaskInstance

**Figura 54:** Back-end::Rules::Task

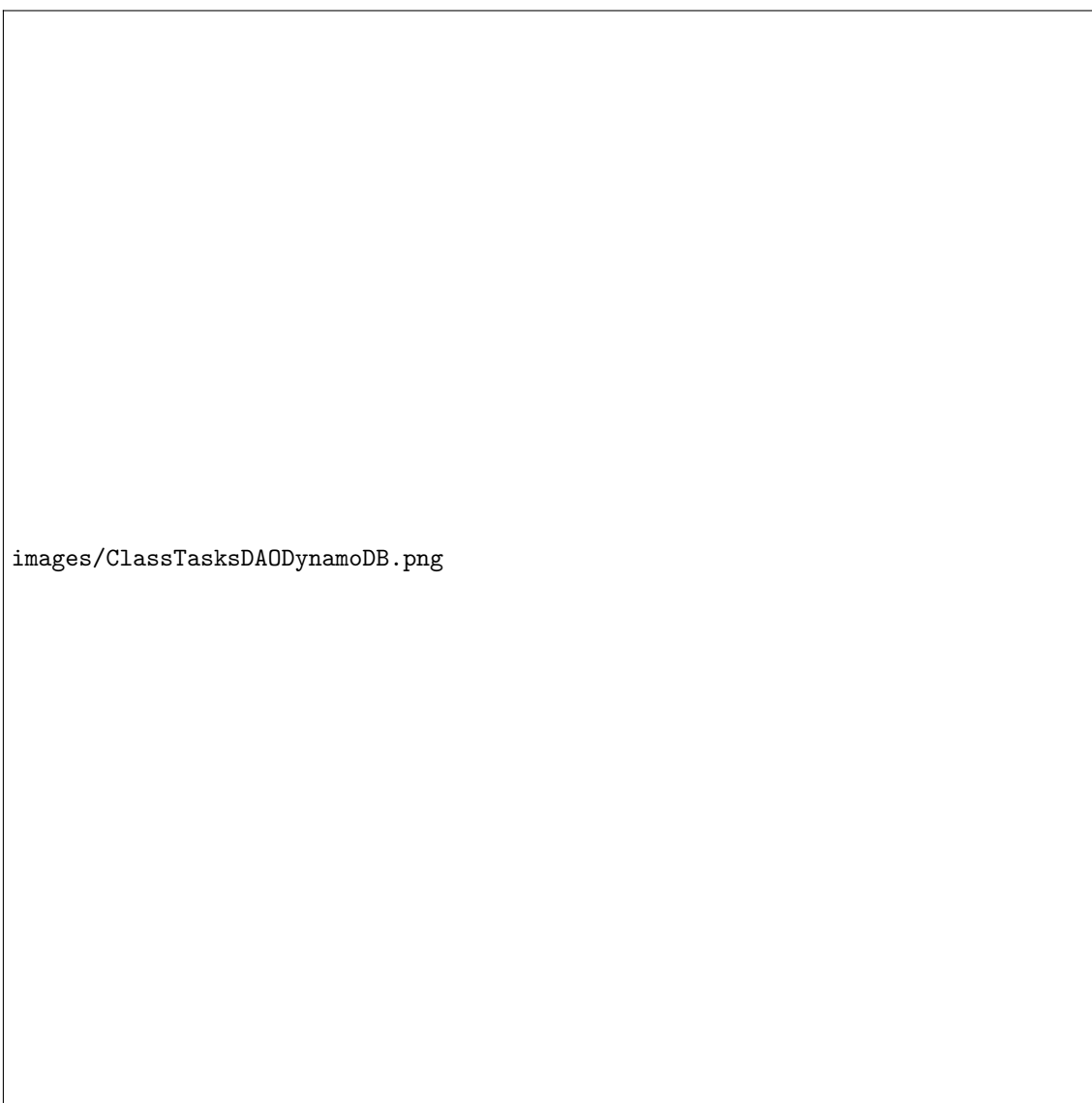
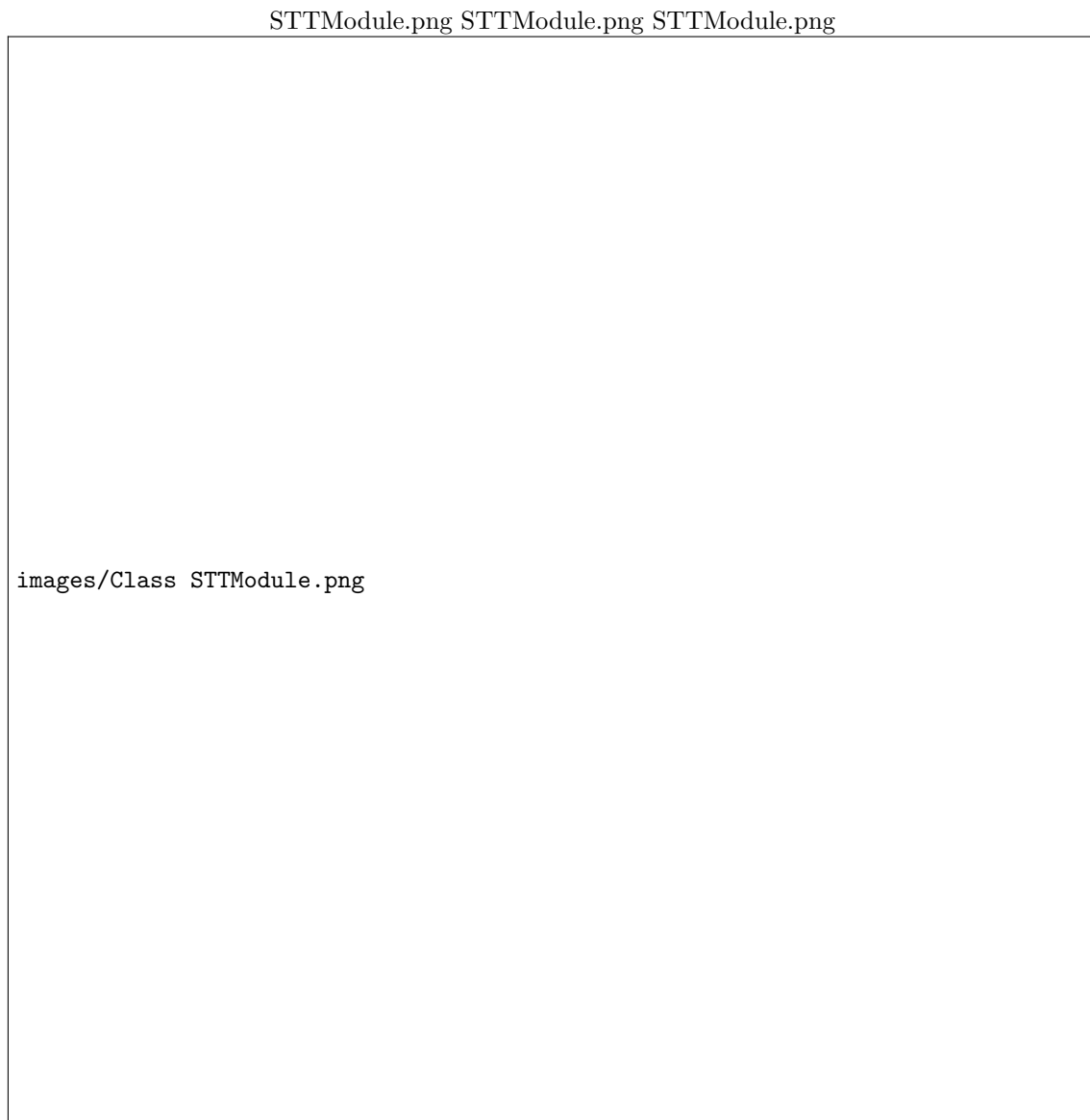
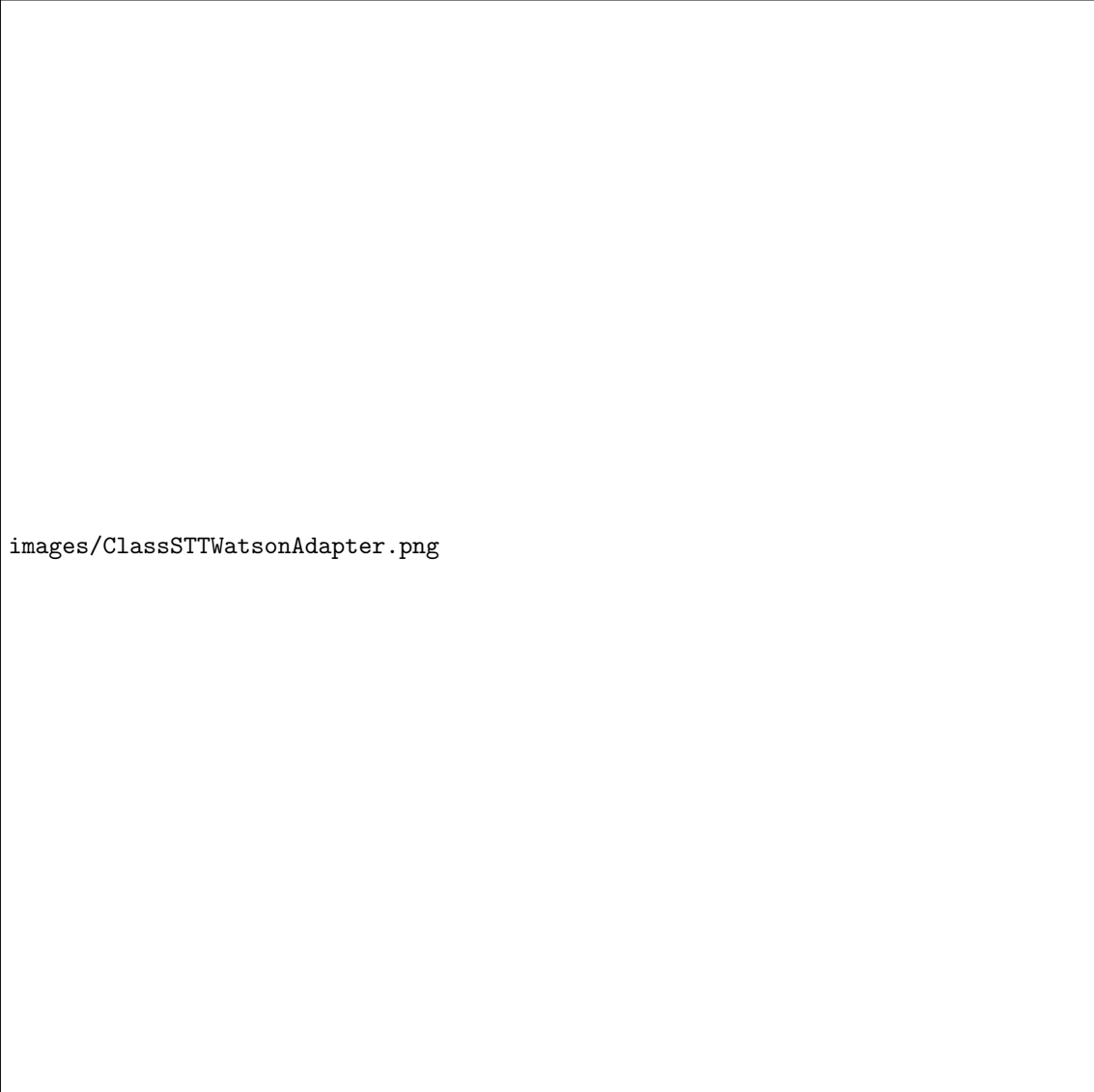


Figura 55: Back-end::Rules::TasksDAODynamoDB

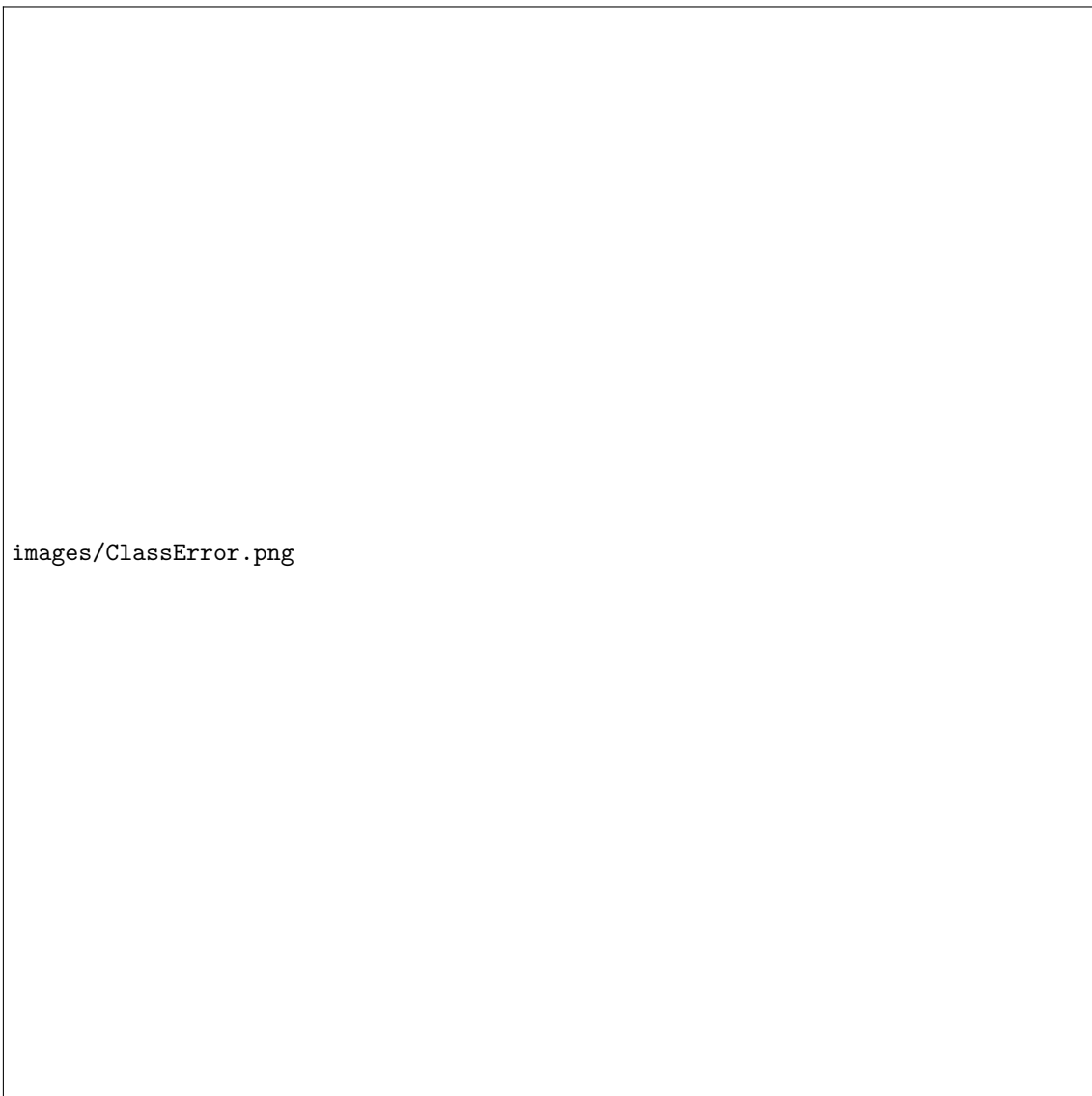
**Figura 56:** Back-end::STT:: STTModule

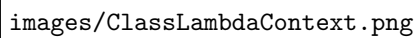


The diagram area is mostly blank, with the text `images/ClassSTTWatsonAdapter.png` appearing on the left side. This likely indicates that the diagram content is missing or has failed to load from the specified file path.

`images/ClassSTTWatsonAdapter.png`

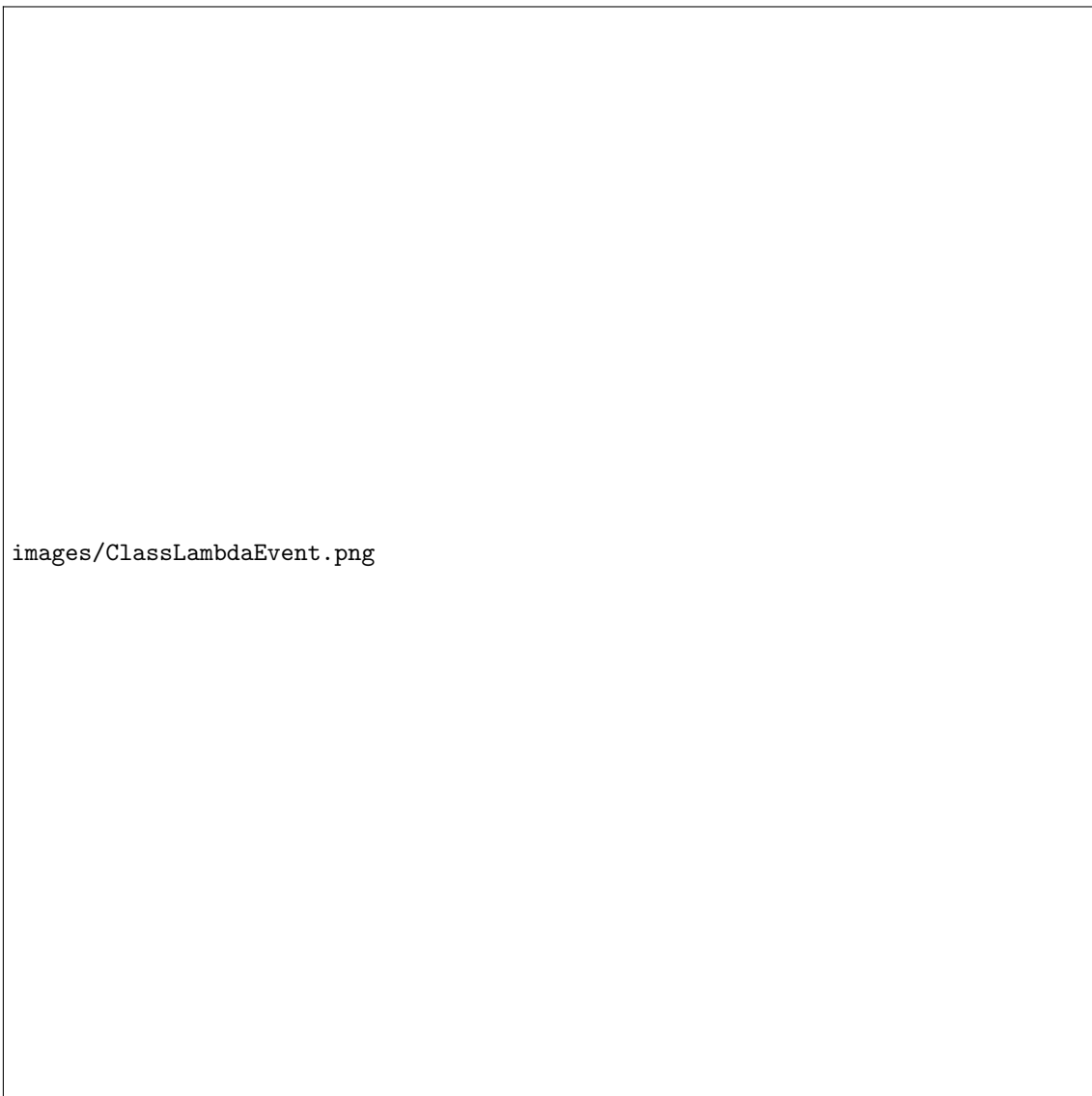
Figura 57: Back-end::STT::STTWatsonAdapter

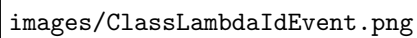
**Figura 58:** Back-end::Utility::Error



images/ClassLambdaContext.png

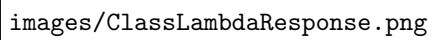
Figura 59: Back-end::Utility::LambdaContext

**Figura 60:** Back-end::Utility::LambdaEvent



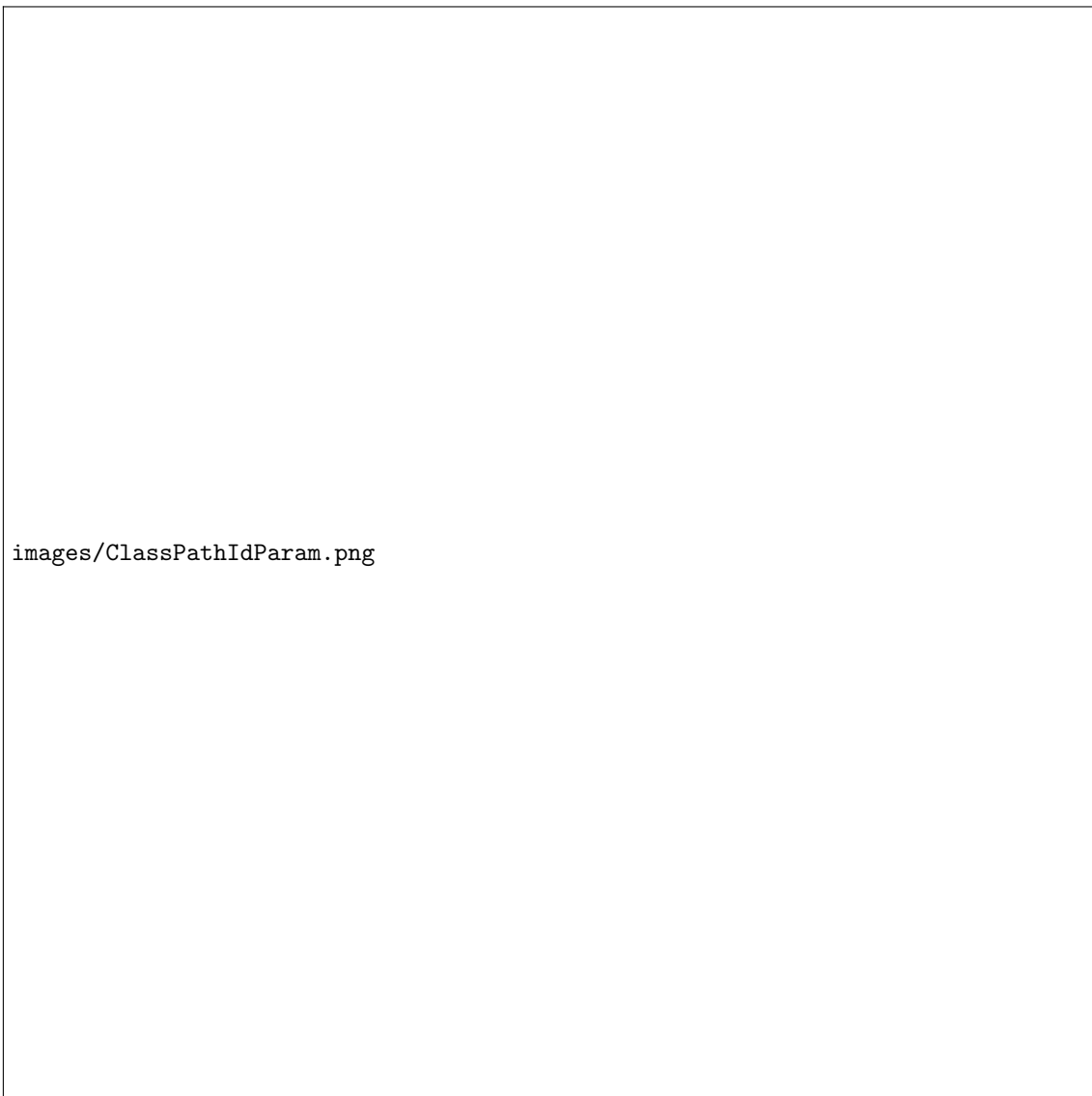
images/ClassLambdaIdEvent.png

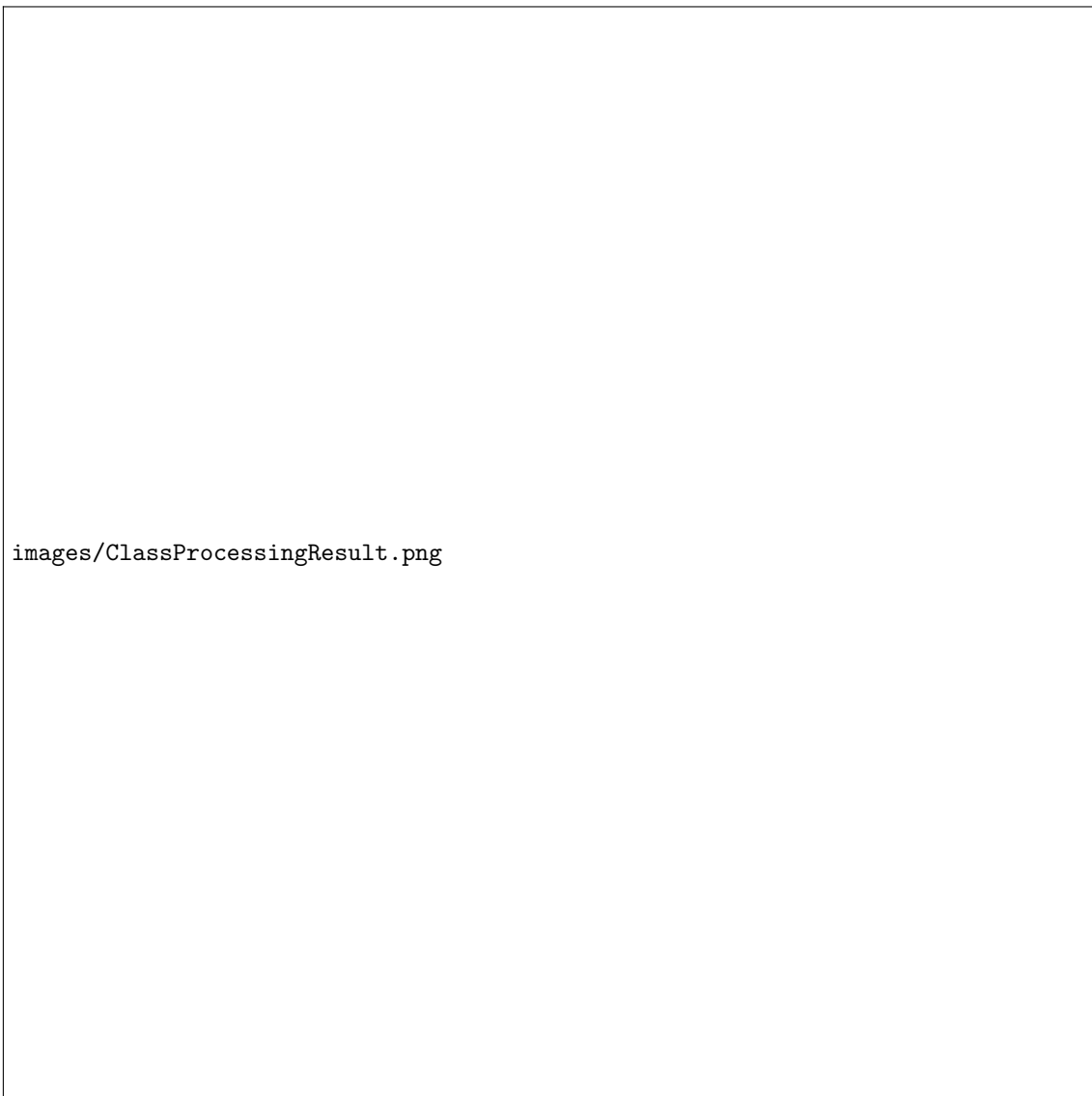
Figura 61: Back-end::Utility::LambdaIdEvent



images/ClassLambdaResponse.png

Figura 62: Back-end::Utility::LambdaResponse

**Figura 63:** Back-end::Utility::PathIdParam

**Figura 64:** Back-end::Utility::ProcessingResult

**Figura 65:** Back-end::Utility::StatusObject

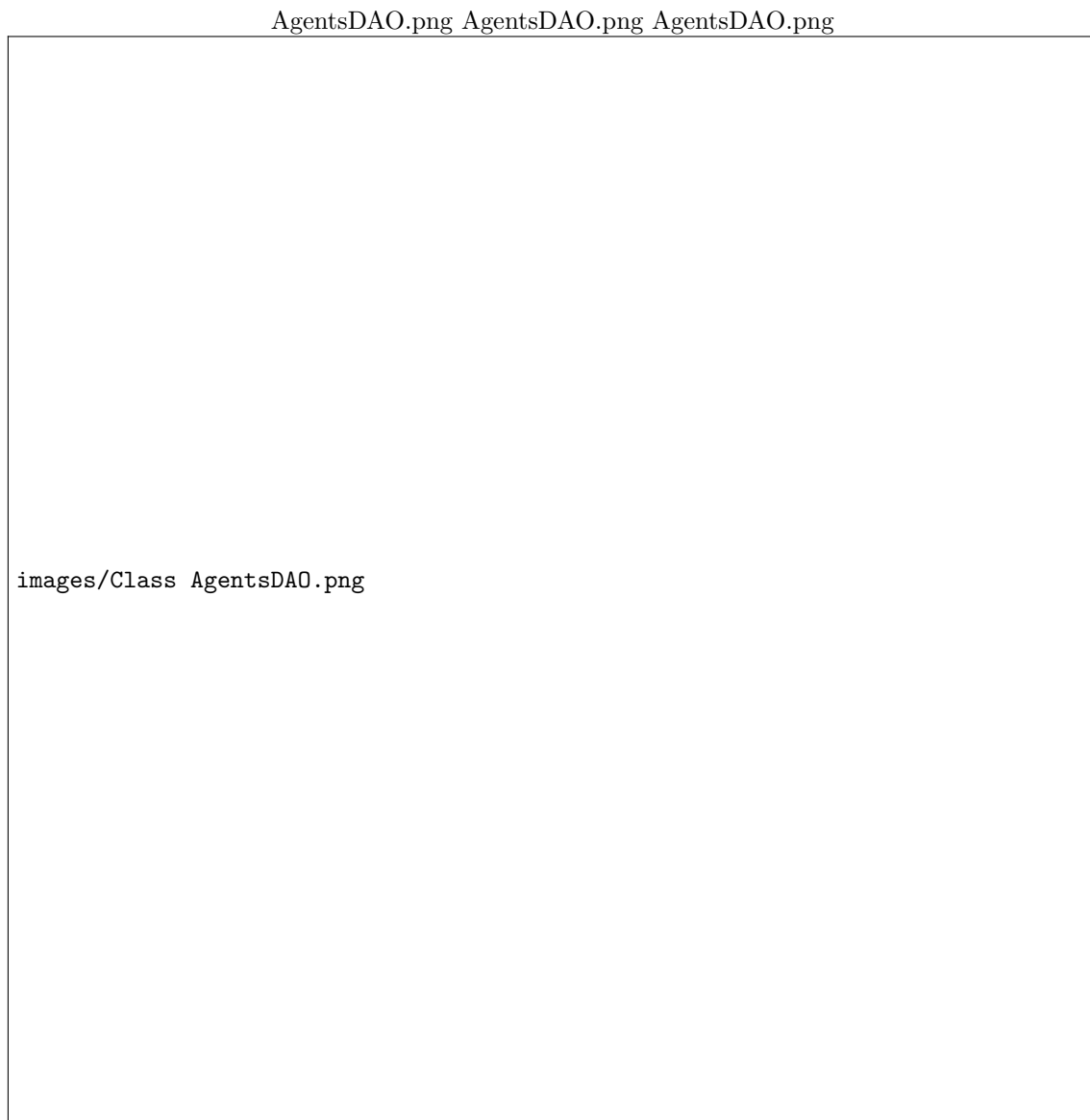
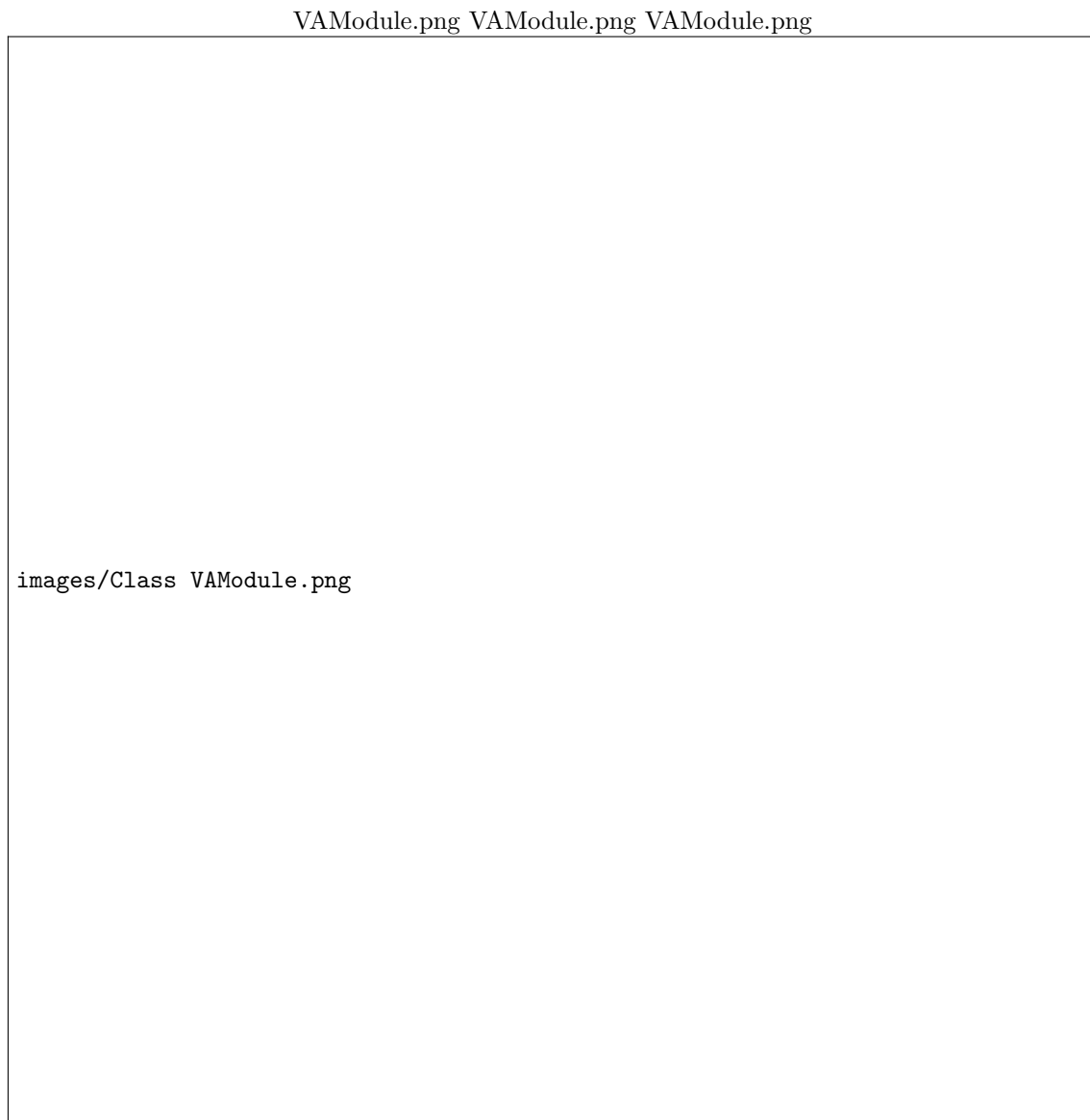
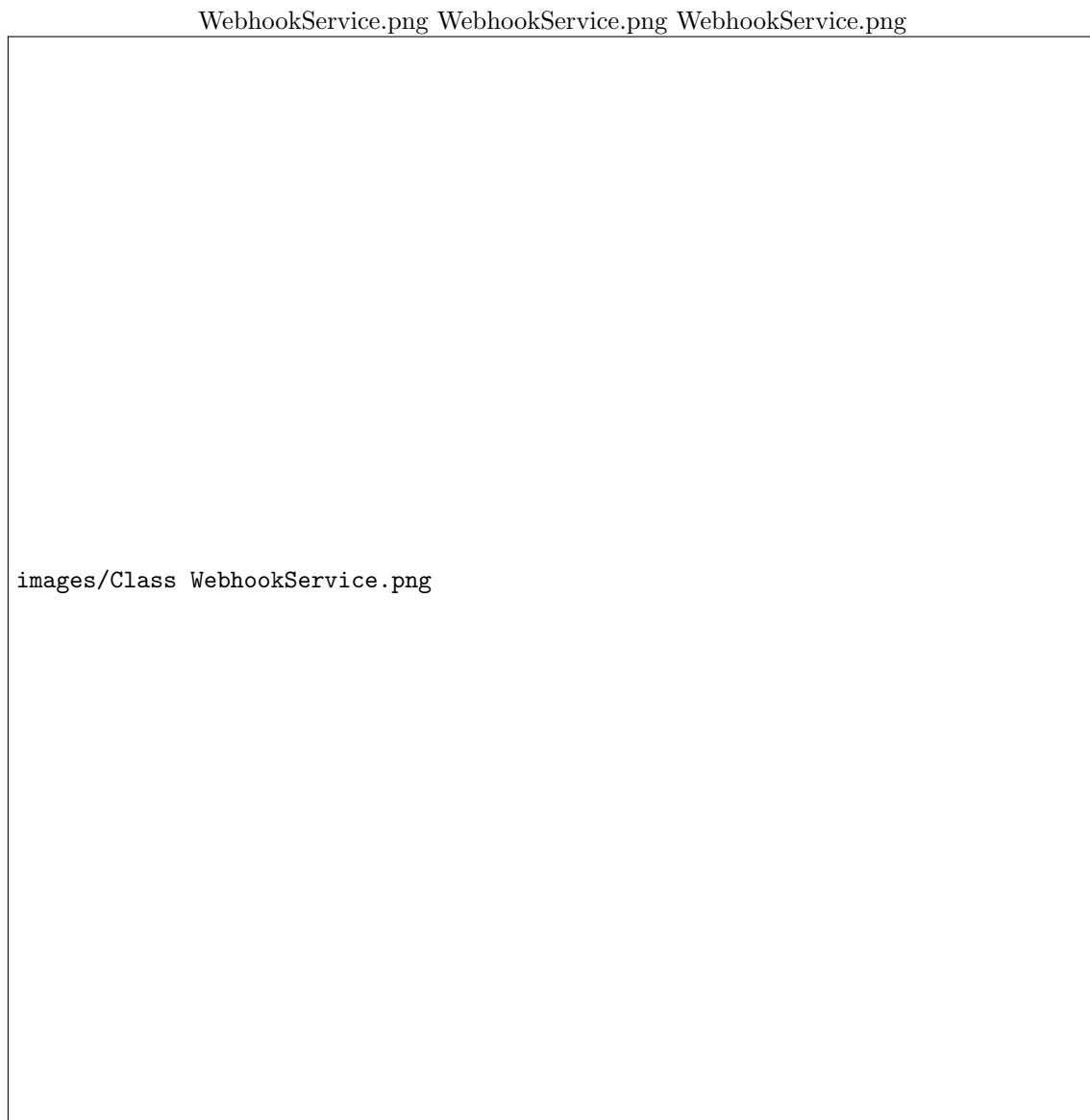


Figura 66: Back-end::VirtualAssistant:: AgentsDAO

**Figura 67:** Back-end::VirtualAssistant:: VAModule

**Figura 68:** Back-end::VirtualAssistant:: WebhookService

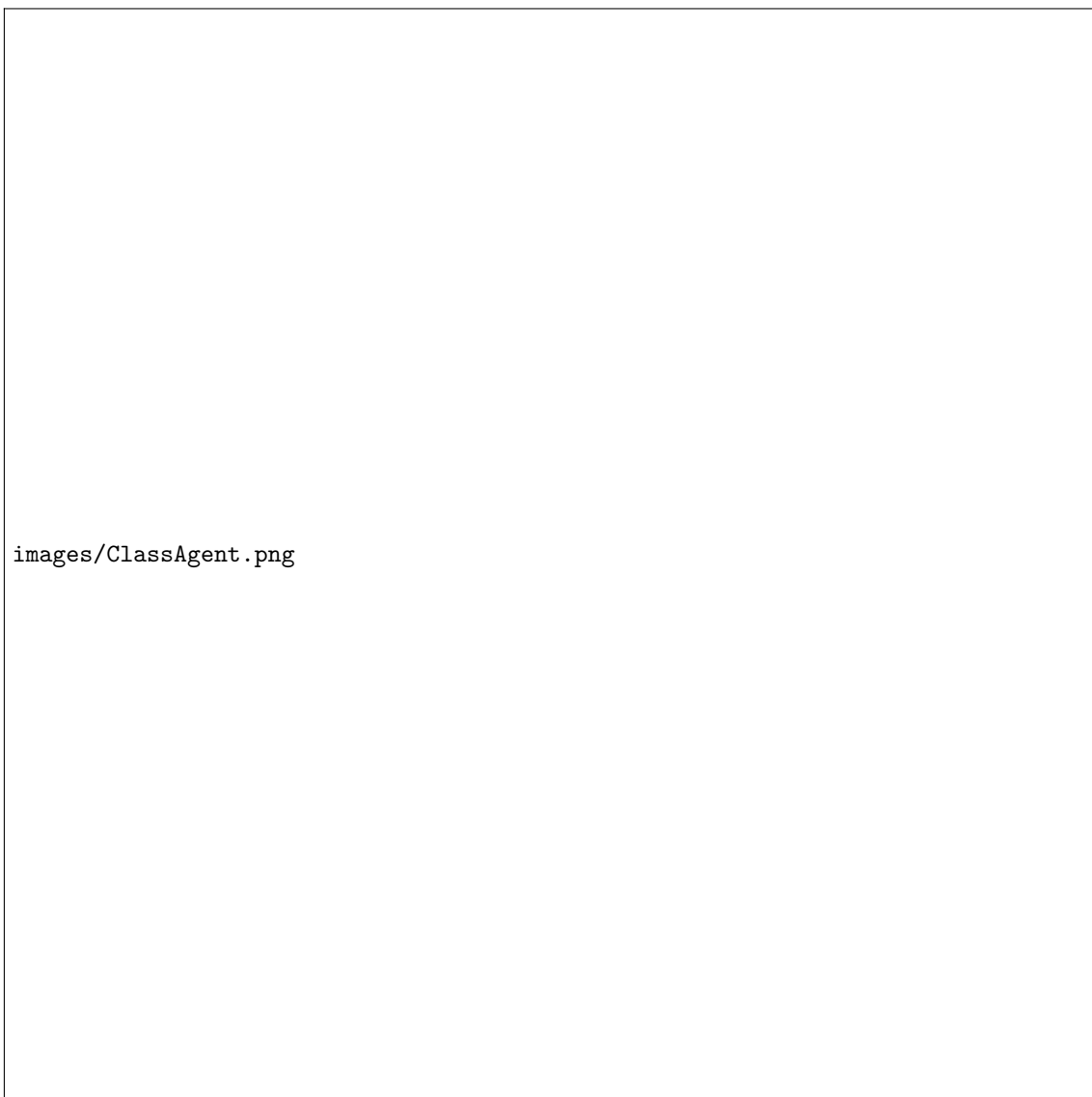


Figura 69: Back-end::VirtualAssistant::Agent

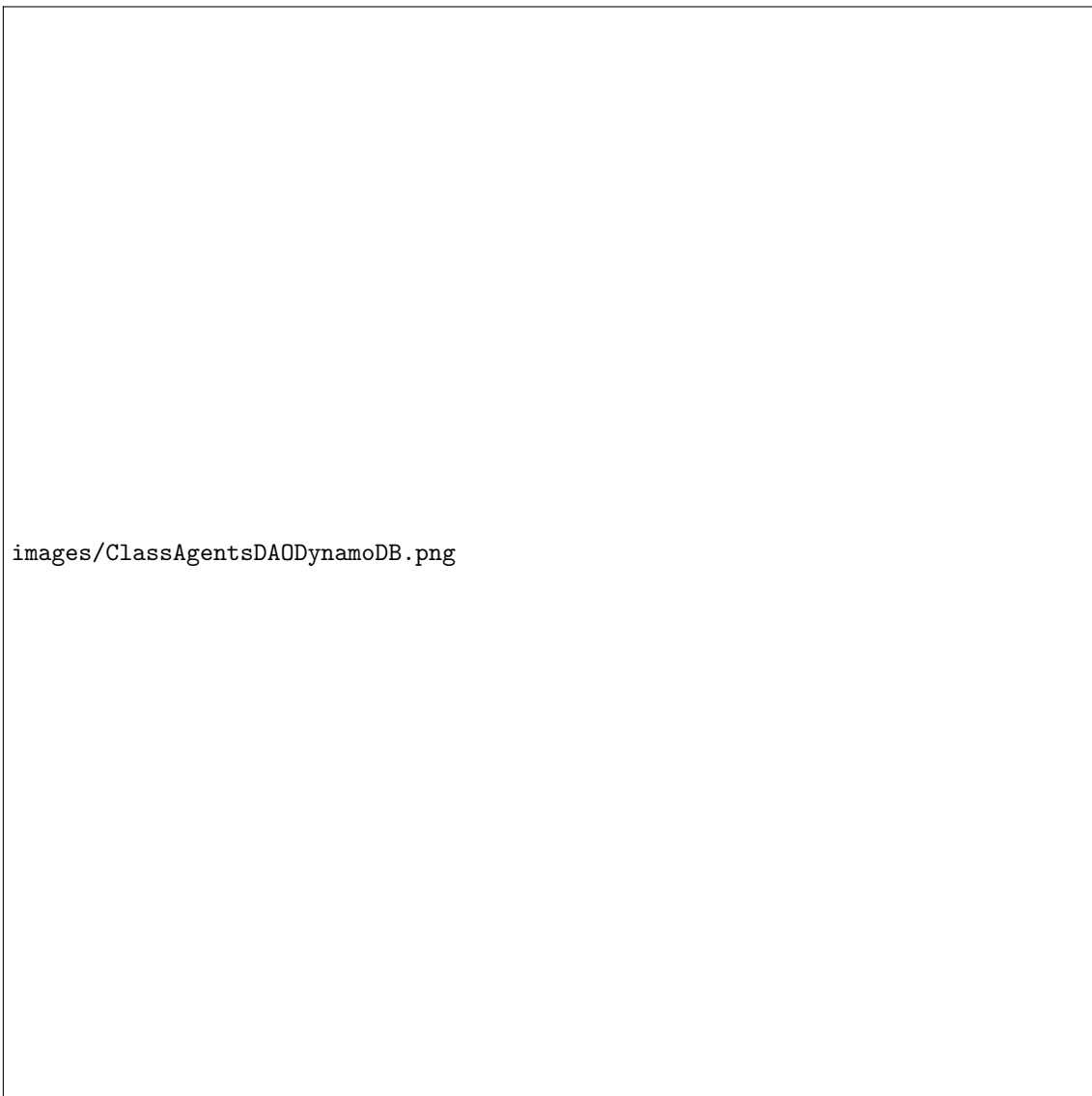
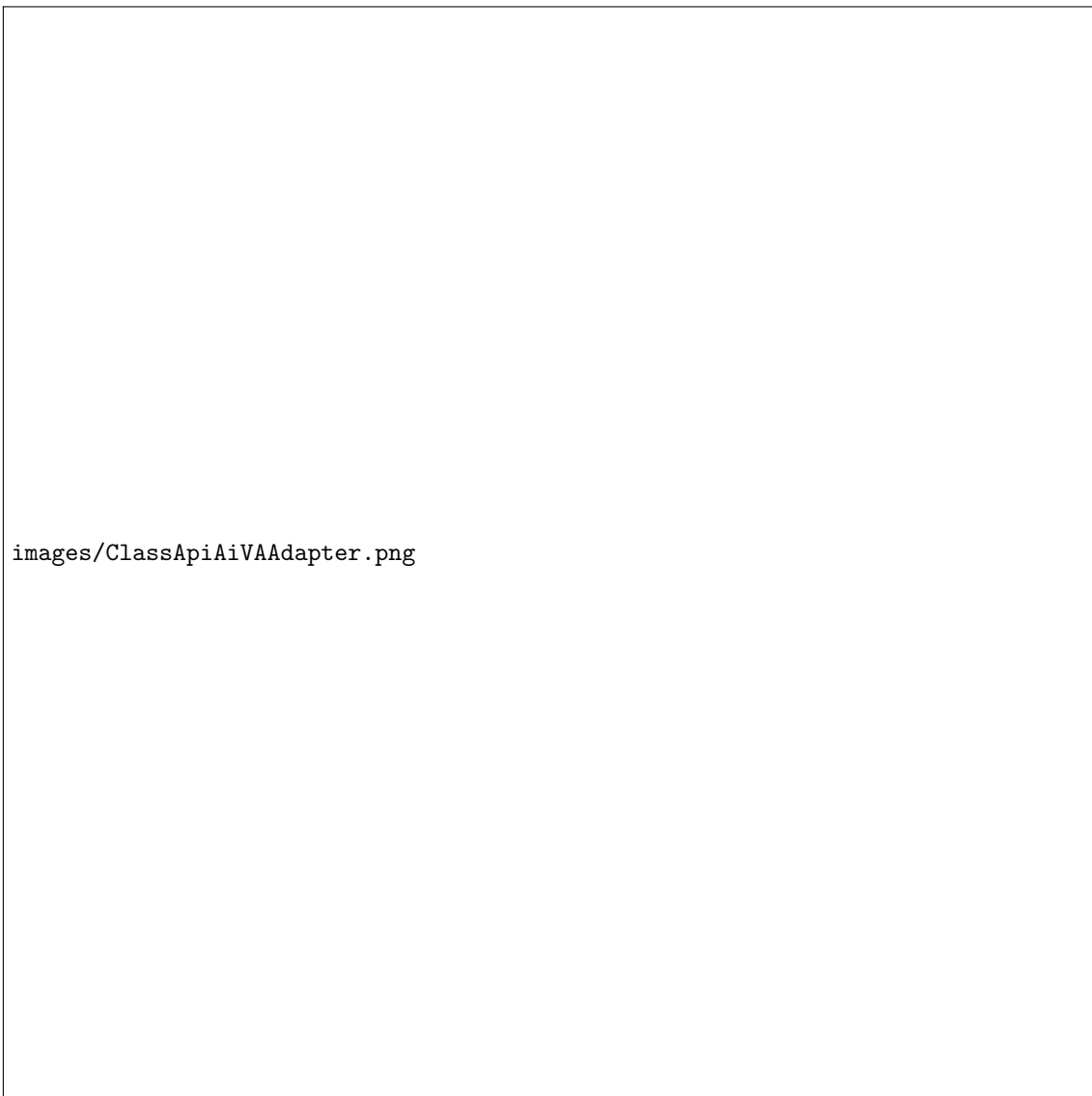


Figura 70: Back-end::VirtualAssistant::AgentsDAODynamoDB

**Figura 71:** Back-end::VirtualAssistant::ApiAiVAAdapter

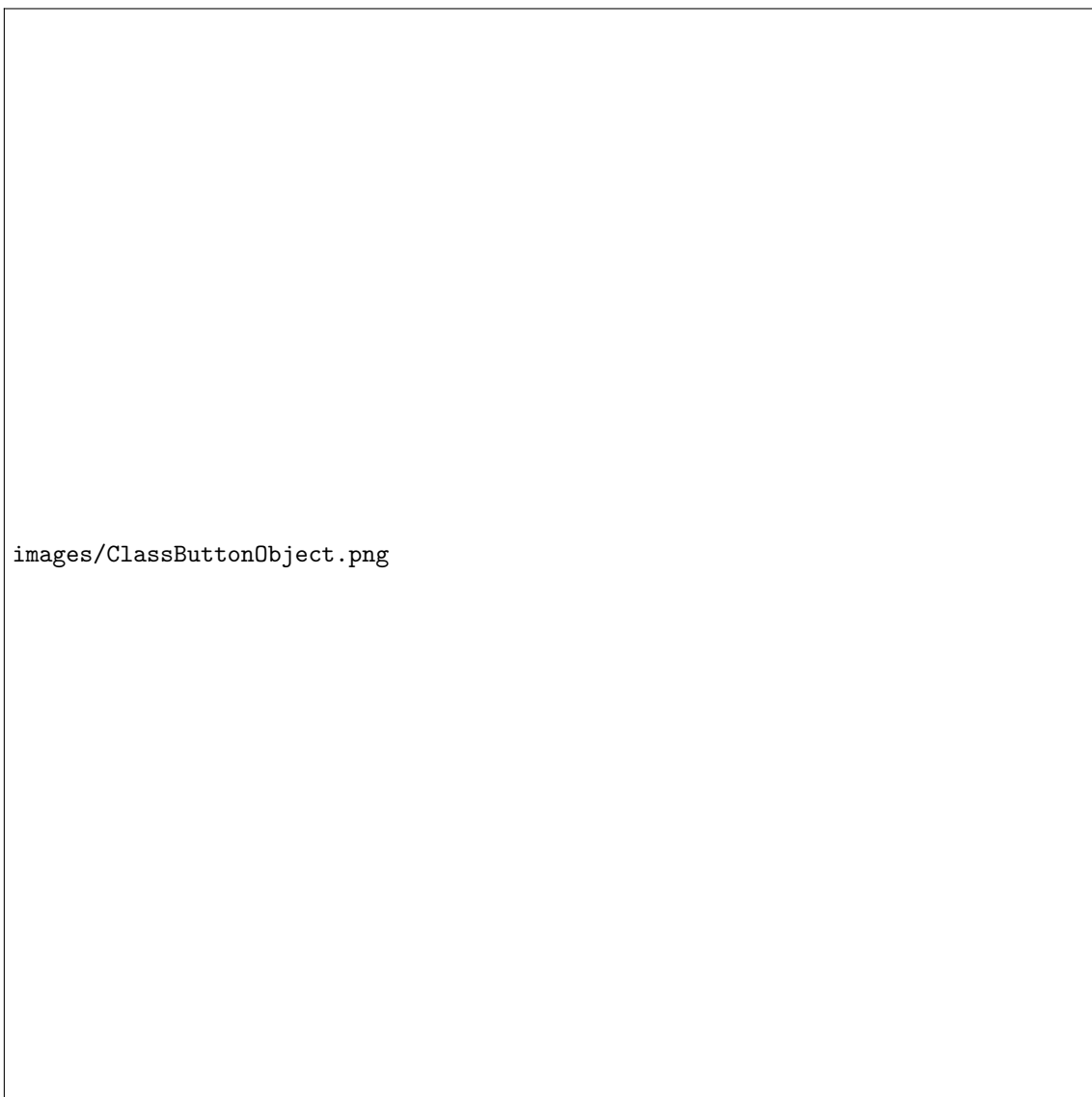


Figura 72: Back-end::VirtualAssistant::ButtonObject

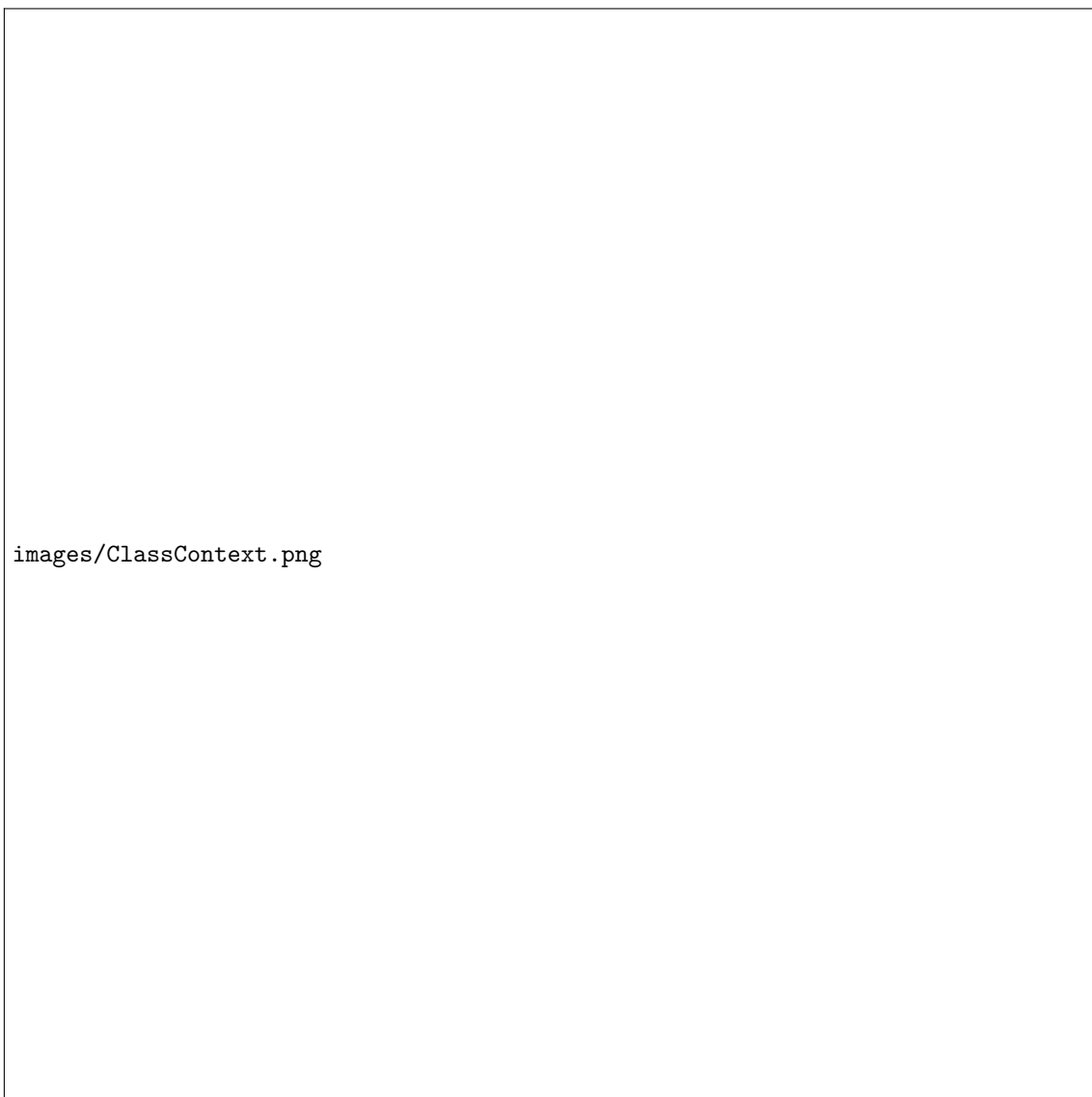
**Figura 73:** Back-end::VirtualAssistant::Context



Figura 74: Back-end::VirtualAssistant::Fulfillment

**Figura 75:** Back-end::VirtualAssistant::Metadata

**Figura 76:** Back-end::VirtualAssistant::MsgObject

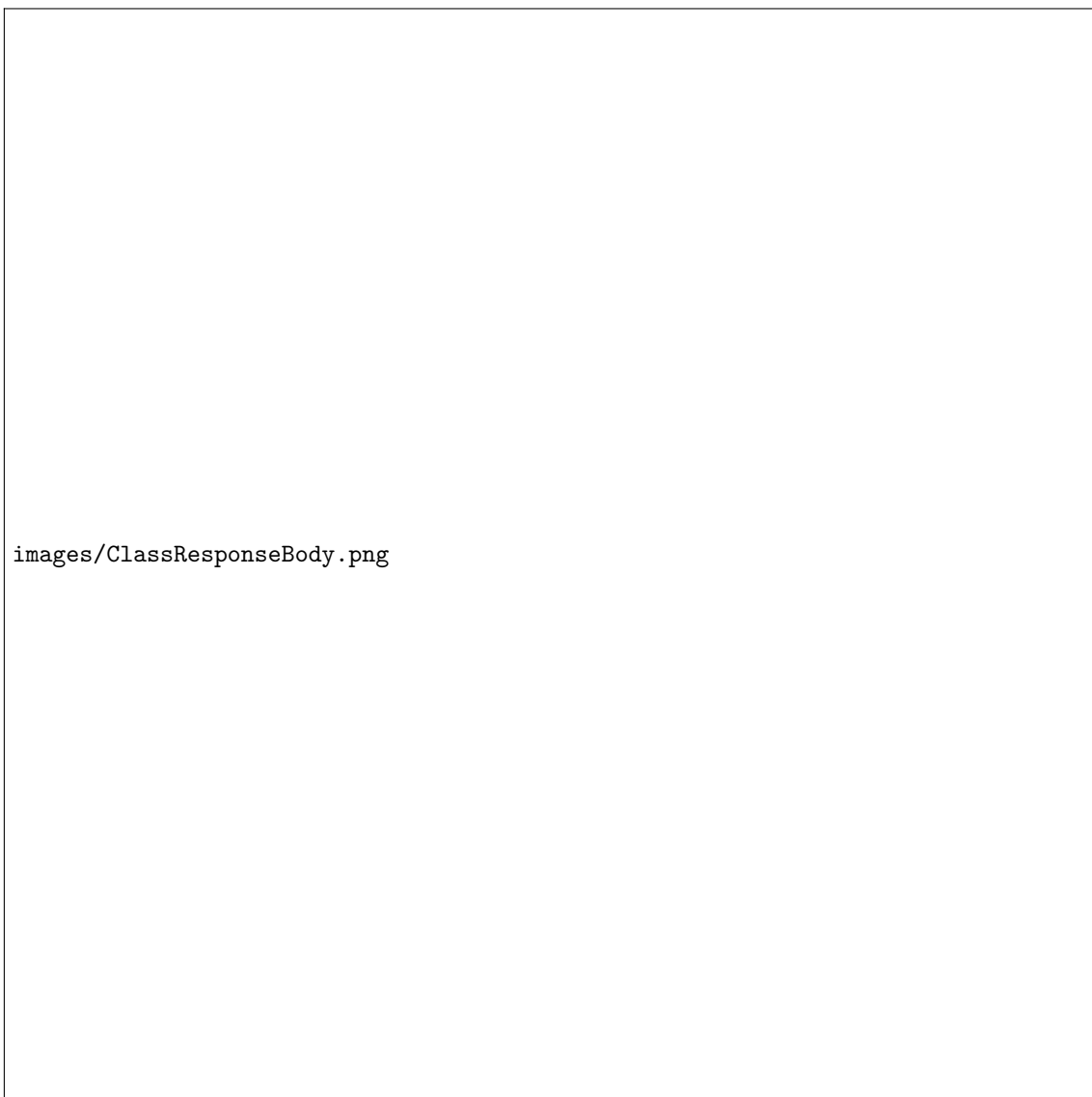
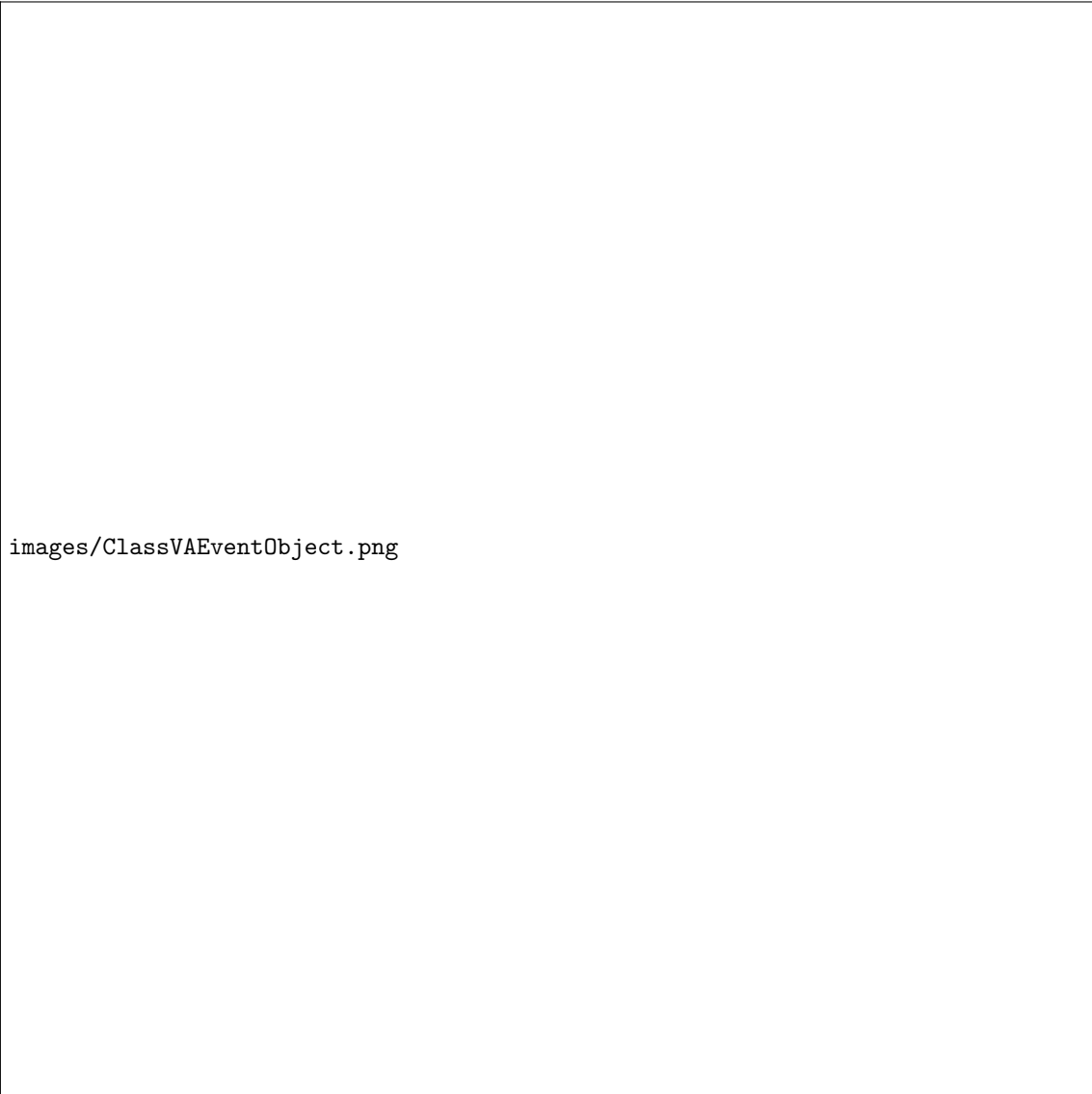


Figura 77: Back-end::VirtualAssistant::ResponseBody

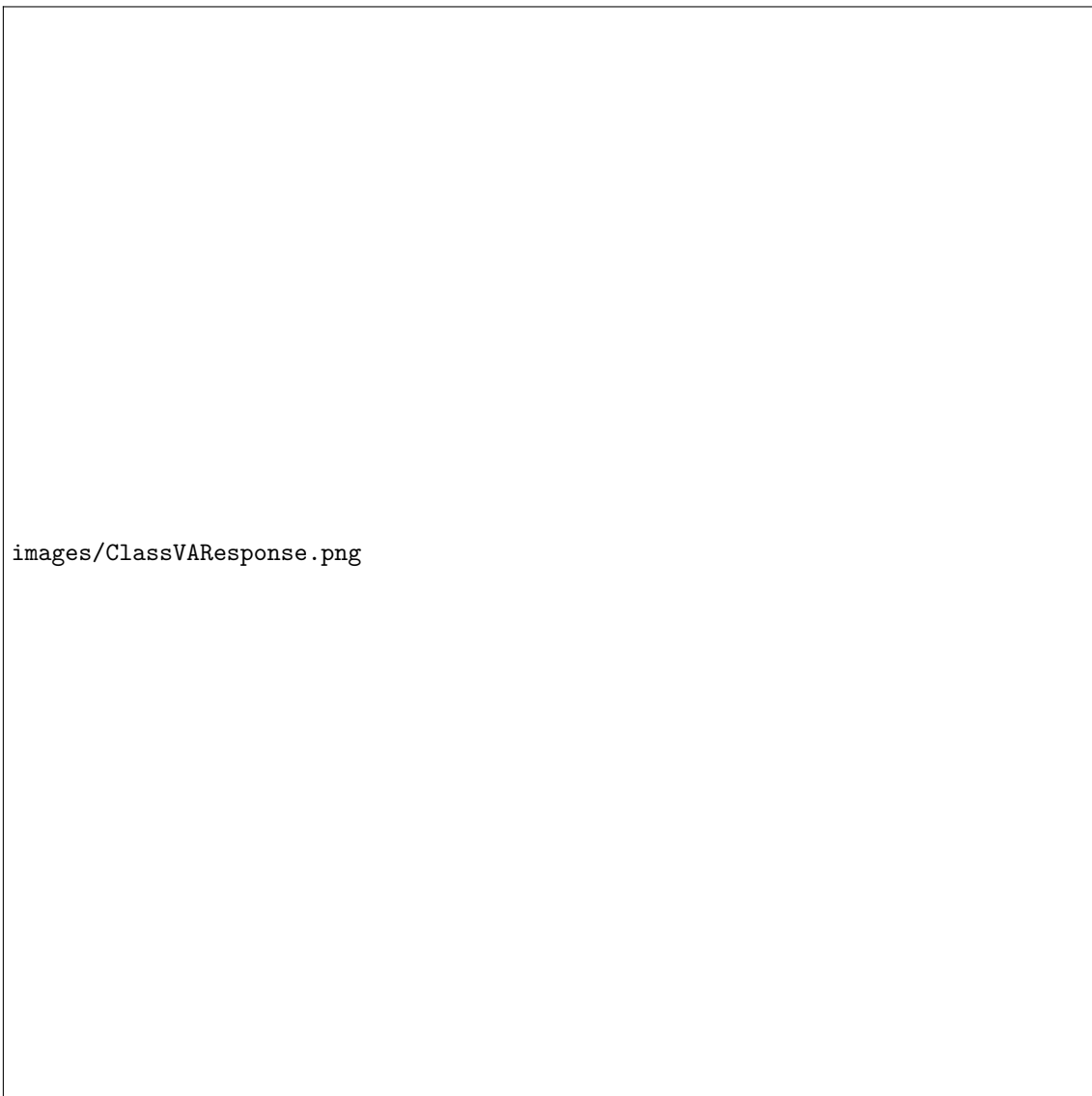


images/ClassVAEventObject.png

Figura 78: Back-end::VirtualAssistant::VAEventObject



Figura 79: Back-end::VirtualAssistant::VAQuery

**Figura 80:** Back-end::VirtualAssistant::VAResponse

**Figura 81:** Back-end::VirtualAssistant::VAService

Tracciamento

Tracciamento Classi-Requisiti

Classe	Requisiti
Back-end::APIGateway::VocalAPI	RFO1
Back-end::Auth:::<<interface>> UsersDAO	RFO2.2 RFO2.2.1
Back-end::Auth:::SRUser	RFO1 RFO1.1.2 RFO1.1.2.1 RFF1.1.2.2 RFO2
Back-end::Auth:::User	RFO1 RFO1.1.2 RFO1.1.2.1 RFF1.1.2.2 RFF1.2
Back-end::Auth:::UsersService	RFD9 RFD9.1 RFD9.1.1 RFD9.1.1.1 RFF9.1.1.2 RFD9.1.3
Back-end::Auth:::VocalLoginModule	RFO1 RFO1.1.2 RFO1.1.2.1
Back-end::Notifications:::Action	RFO13 RFO13.1 RFD13.2
Back-end::Notifications:::Attachment	RFO13 RFO13.1 RFD13.2
Back-end::Notifications:::- NotificationChannel	RFO13 RFO13.1 RFD13.2
Back-end::Notifications:::- NotificationMessage	RFO13 RFO13.1 RFD13.2
Back-end::Notifications:::Purpose	RFO13 RFO13.1 RFD13.2
Back-end::Notifications:::Topic	RFO13 RFO13.1 RFD13.2
Back-end::Rules:::Task	RFO2.1 RFO2.1.1 RFO2.1.1.1 RFO2.1.1.2 RFO2.1.1.3 RFD2.1.1.4 RFD2.1.1.4.1

Classe	Requisiti
	RFO2.1.1.5 RFO2.1.2 RFD2.1.3 RFD2.1.3.1 RFD2.1.3.2 RFD2.1.3.2.1 RFD2.1.3.2.2 RFD2.1.3.3 RFD2.1.3.4 RFD2.1.3.5 RFD2.1.3.5.1 RFD2.1.3.5.2 RFD2.1.3.6
Client::ApplicationManager::~- ApplicationPackage	RVO9
Client::Logic::LogicObserver	RFO1 RFO1.1 RFO1.1.1 RFO1.1.1.1 RFO1.1.1.2 RFO1.1.2 RFO1.1.2.1 RFF1.1.2.2 RFO1.1.3 RFO2 RFO2.1 RFO2.1.1 RFO2.1.1.1 RFO2.1.1.1.1 RFO2.1.1.2 RFO2.1.1.3 RFO2.1.1.3.1 RFO2.1.1.3.2 RFO2.1.1.3.3 RFD2.1.1.4 RFD2.1.1.4.1 RFO2.1.1.5 RFO2.1.2 RFO2.1.2.1 RFD2.1.3 RFD2.1.3.1 RFD2.1.3.2 RFD2.1.3.2.1 RFD2.1.3.2.2 RFD2.1.3.3 RFD2.1.3.4 RFD2.1.3.5 RFD2.1.3.5.1 RFD2.1.3.5.2 RFD2.1.3.5.3 RFD2.1.3.5.4 RFD2.1.3.6
Client::Recorder::Recorder	RFO3 RFO3.1

Classe	Requisiti
	RFD3.2 RFD3.2.1 RFD3.2.2 RFD3.2.3 RFD3.2.3.1

Tabella 1: Tracciamento Classi-Requisiti

Tracciamento Componenti-Requisiti

Componente	Requisiti
Back-end	RFO1 RFO1.1.2 RFO1.1.2.1 RFF1.1.2.2 RFF1.2 RFO2 RFO2.1 RFO2.1.1 RFO2.1.1.1 RFO2.1.1.2 RFO2.1.1.3 RFD2.1.1.4 RFD2.1.1.4.1 RFO2.1.1.5 RFO2.1.2 RFD2.1.3 RFD2.1.3.1 RFD2.1.3.2 RFD2.1.3.2.1 RFD2.1.3.2.2 RFD2.1.3.3 RFD2.1.3.4 RFD2.1.3.5 RFD2.1.3.5.1 RFD2.1.3.5.2 RFD2.1.3.6 RFO2.2 RFO2.2.1 RFD9 RFD9.1 RFD9.1.1 RFD9.1.1.1 RFF9.1.1.2 RFD9.1.3 RFO13 RFO13.1 RFD13.2
Back-end::APIGateway	RFO1
Back-end::Auth	RFO1 RFO1.1.2 RFO1.1.2.1 RFF1.1.2.2 RFF1.2 RFO2 RFO2.2 RFO2.2.1 RFD9 RFD9.1 RFD9.1.1 RFD9.1.1.1 RFF9.1.1.2 RFD9.1.3

Componente	Requisiti
Back-end::Notifications	RFO13 RFO13.1 RFD13.2
Back-end::Rules	RFO2.1 RFO2.1.1 RFO2.1.1.1 RFO2.1.1.2 RFO2.1.1.3 RFD2.1.1.4 RFD2.1.1.4.1 RFO2.1.1.5 RFO2.1.2 RFD2.1.3 RFD2.1.3.1 RFD2.1.3.2 RFD2.1.3.2.1 RFD2.1.3.2.2 RFD2.1.3.3 RFD2.1.3.4 RFD2.1.3.5 RFD2.1.3.5.1 RFD2.1.3.5.2 RFD2.1.3.6
Client	RFO1 RFO1.1 RFO1.1.1 RFO1.1.1.1 RFO1.1.1.2 RFO1.1.2 RFO1.1.2.1 RFF1.1.2.2 RFO1.1.3 RFO2 RFO2.1 RFO2.1.1 RFO2.1.1.1 RFO2.1.1.1.1 RFO2.1.1.2 RFO2.1.1.3 RFO2.1.1.3.1 RFO2.1.1.3.2 RFO2.1.1.3.3 RFD2.1.1.4 RFD2.1.1.4.1 RFO2.1.1.5 RFO2.1.2 RFO2.1.2.1 RFD2.1.3 RFD2.1.3.1 RFD2.1.3.2 RFD2.1.3.2.1 RFD2.1.3.2.2 RFD2.1.3.3 RFD2.1.3.4

Componente	Requisiti
	RFD2.1.3.5 RFD2.1.3.5.1 RFD2.1.3.5.2 RFD2.1.3.5.3 RFD2.1.3.5.4 RFD2.1.3.6 RFO3 RFO3.1 RFD3.2 RFD3.2.1 RFD3.2.2 RFD3.2.3 RFD3.2.3.1 RVO9
Client::ApplicationManager	RVO9
Client::Logic	RFO1 RFO1.1 RFO1.1.1 RFO1.1.1.1 RFO1.1.1.2 RFO1.1.2 RFO1.1.2.1 RFF1.1.2.2 RFO1.1.3 RFO2 RFO2.1 RFO2.1.1 RFO2.1.1.1 RFO2.1.1.1.1 RFO2.1.1.2 RFO2.1.1.3 RFO2.1.1.3.1 RFO2.1.1.3.2 RFO2.1.1.3.3 RFD2.1.1.4 RFD2.1.1.4.1 RFO2.1.1.5 RFO2.1.2 RFO2.1.2.1 RFD2.1.3 RFD2.1.3.1 RFD2.1.3.2 RFD2.1.3.2.1 RFD2.1.3.2.2 RFD2.1.3.3 RFD2.1.3.4 RFD2.1.3.5 RFD2.1.3.5.1 RFD2.1.3.5.2 RFD2.1.3.5.3 RFD2.1.3.5.4 RFD2.1.3.6
Client::Recorder	RFO3 RFO3.1

Componente	Requisiti
	RFD3.2
	RFD3.2.1
	RFD3.2.2
	RFD3.2.3
	RFD3.2.3.1

Tabella 2: Tracciamento Componenti-Requisiti

Design Patterns

Architetturali

Architettura a microservizi

- **Scopo:** l'architettura a microservizi è un approccio allo sviluppo di una singola applicazione come insieme di piccoli servizi, ciascuno dei quali viene eseguito da un proprio processo e comunica con un meccanismo snello, spesso una HTTP API;
- **Vantaggi:**
 - ogni microservizio è relativamente piccolo, quindi più semplice da implementare e da capire per gli sviluppatori;
 - ogni microservizio è indipendente dagli altri; è quindi possibile distribuire nuove versioni più frequentemente e isolare i possibili errori.
- **Svantaggi:**
 - l'architettura risulta maggiormente complessa perchè risulta essere un sistema distribuito;
 - la gestione di più microservizi potrebbe risultare in un carico di lavoro maggiore rispetto ad una sua versione monolitica.
- **Utilizzo:**

Architettura event-driven

- **Scopo:** anche se non è un vero e proprio pattern, l'architettura event-driven è un particolare tipo di architettura asincrona per sistemi distribuiti basata sugli eventi.
- **Vantaggi:**
 - per definizione, questo tipo di architettura è particolarmente adatto ad ambienti di tipo asincrono basati sugli eventi, come ad esempio l'interazione con degli utenti in tempo reale.
- **Svantaggi:**
 - i sistemi che utilizzano tale architettura sono spesso distribuiti: ciò comporta un maggiore livello di complessità.
- **Utilizzo:**

Client-side discovery

- **Scopo:** all'interno di un'architettura a microservizi, i singoli microservizi si trovano spesso in posizioni non fissate in quanto decise dinamicamente. Un metodo per la loro localizzazione consiste nel pattern Client-side discovery, che consiste nella richiesta della posizione di uno specifico microservizio da parte del client ad un registro, che conosce le posizioni di tutte le istanze dei microservizi.
- **Vantaggi:**
 - permette di allocare dinamicamente diverse istanze di diversi servizi.
- **Svantaggi:**
 - crea dipendenze tra il registro e il client.

- **Utilizzo:**

Data Access Object

- **Scopo:** il pattern Data Access Object (DAO) consiste nell'utilizzo di un oggetto che fornisce un'interfaccia astratta per la gestione di un database, o più in generale per la gestione della persistenza.
- **Vantaggi:**
 - separazione tra logica di business e dati;
 - modifiche sui dati non comportano modifiche sul client che utilizza DAO.
- **Svantaggi:**
 - un'interfaccia di questo tipo potrebbe nascondere i costi di accesso ad un database;
 - potrebbe essere necessarie molte più operazioni rispetto all'esecuzione diretta di una query su un database.
- **Utilizzo:**

Dependency Injection

- **Scopo:** consiste nella separazione del comportamento di una componente dalla risoluzione delle sue dipendenze.
- **Vantaggi:**
 - la separazione del comportamento dalle dipendenze rende una componente molto più flessibile;
 - rende le singole componenti maggiormente indipendenti permettendo una più facile progettazione dei test di unità.
- **Svantaggi:**
 - eventuali errori legati alla risoluzione delle dipendenze o alla loro implementazione vengono rilevati solamente a runtime;
 - rende più difficile il tracciamento del codice in quanto ne separa la costruzione dal comportamento.
- **Utilizzo:**

Strutturali

Facade

- **Scopo:** indica un oggetto che permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro, nonché a blocchi di codice complessi.
- **Vantaggi:**
 - permette di nascondere la complessità di un'operazione: rispetto alla chiamata diretta di un sottoinsieme di classi è possibile chiamare solamente la classe definita come facade semplificando l'operazione;
 - permette di diminuire le dipendenze tra sottosistemi;

- **Svantaggi:**
 - i sottosistemi risultano essere collegati al facade: modifiche alla struttura dei sottosistemi comportano una serie di modifiche al facade stesso;
- **Utilizzo:**

Adapter

- **Scopo:** questo pattern permette la comunicazione tra due interfacce completamente differenti tramite l'utilizzo di un Adapter.
- **Vantaggi:**
 - permette la conversione di una classe esistente in un'altra completamente differente senza modificarne il codice;
 - maggiore flessibilità nella progettazione.
- **Svantaggi:**
 - aumenta la dimensione del codice;
 - a volte per interconnettere due interfacce sono necessari più Adapter.
- **Utilizzo:**

Creazionali

Singleton

- **Scopo:** questo pattern ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale ad essa.
- **Vantaggi:**
 - questo pattern risulta molto utile ogni qual volta è necessaria una sola istanza di una classe.
- **Svantaggi:**
 - la classe Singleton risulta essere globale, e di conseguenza rende più difficile la definizione di test di unità;
 - aumenta il livello di accoppiamento del codice.
- **Utilizzo:**

Module

- **Scopo:** questo pattern ha lo scopo di introdurre il concetto di modularità nel linguaggi di programmazione che non lo possiedono.
- **Vantaggi:**
 - come da definizione, questo pattern permette l'implementazione della modularità in ambienti privi di supporto ad essa.
- **Svantaggi:**
 - la sua implementazione richiede un maggiore carico di lavoro.
- **Utilizzo:**

Comportamentali

Observer

- **Scopo:** questo pattern permette la definizione di una o più classi Observer le quali "osservano" una classe Soggetto e ne gestiscono gli eventi.
- **Vantaggi:**
 - permette la gestione di eventi tramite l'invio di dati ad altre classi in modo efficiente;
 - la definizione di classi Observer non causa modifiche alla classe Soggetto.
- **Svantaggi:**
 - una cattiva implementazione comporta un aumento della complessità del codice;
 - l'interfaccia Observer deve essere implementata, e ciò comporta ereditarietà.
- **Utilizzo:**

Tecnologie utilizzate

Promise e Observable

JavaScript è un linguaggio single-thread, quindi basato su un singolo thread in esecuzione. Questo significherebbe dover aspettare sempre il termine di un'operazione prima di passare alla successiva, quindi nel caso di operazioni di lunga durata, il flusso dell'elaborazione principale di un'applicazione JavaScript potrebbe "congelarsi".

Per aggirare questo problema, una delle soluzioni più diffuse è quella di passare una callback alla funzione in questione e, anziché aspettare il compimento dell'operazione, restituisce il controllo al chiamante. Quando l'operazione sarà terminata, la funzione di callback verrà invocata.

L'uso di callback, spesso annidate, rendono il codice di difficile comprensione e di difficile manutenzione. Due possibili soluzioni sono l'uso delle Promise di bluebird e gli Observable di RxJS.

Promise e bluebird

Una Promise è un oggetto che rappresenta il risultato pendente di un'operazione asincrona. Ciò permette ai metodi asincroni di restituire valori alla stessa maniera dei metodi sincroni:

invece del valore finale, il metodo asincrono restituisce una promessa di ottenere un valore in un momento futuro.

I principali metodi di una Promise sono:

- **then:** ritorna una Promise, in questo modo è possibile concatenare successive chiamate a questo metodo. È composto da due parametri opzionali che corrispondono a due funzioni: `onFulfill` viene richiamata se la Promise ha avuto successo, `onReject` se è stata rigettata.

- `catch`: ritorna una Promise e si occupa di gestire eventuali errori generati nella catena dei `then`.

Si è deciso di utilizzare `bluebird` in quanto offre le seguenti caratteristiche:

- `cross-platform`: ideale per progetti che prevedono esperienze multiplatforma;
- `compatibile con le specifiche Promise/A+`: `bluebird` può essere usato come rimpiazzo per Promise nativa offrendo un immediato miglioramento delle prestazioni;
- `debug facile`: gli `stack trace` sono in cui vengono riportati, quando possibile, gli errori non gestiti sono configurabili.

Observable e RxJS v5

Un `Observable` è una rappresentazione di un qualsiasi insieme di valori durante un qualsiasi periodo di tempo. Viene utilizzata per le implementazioni del pattern `Observer`.

`RxJS v5` è ancora in beta ma sostituirà in breve la `v4`. Si è deciso di utilizzarla perchè è una libreria solida nella gestione degli `Observable` e sarà ulteriormente aggiornata

da Microsoft e altri sviluppatori di software open source.

AWS SDK per JavaScript in Node.js

Siccome il gruppo ha deciso di appoggiarsi all'infrastruttura Amazon Web Services (AWS) per lo sviluppo del sistema è necessario utilizzare SDK per JavaScript in Node.js per

interfacciarsi ai singoli servizi offerti da AWS. In particolare si farà utilizzo di `DynamoDb`.

Node.js

La parte Back-End è stata sviluppata tramite la piattaforma event-driven `Node.js`, basata sul motore JavaScript V8. Esso permette di realizzare applicazioni Web utilizzando JavaScript, tipicamente client-side, per la scrittura server-side. La caratteristica principale di `Node.js` risiede nella possibilità che offre di accedere alle risorse del sistema operativo in modalità event-driven non sfruttando il classico modello basato su processi o thread concorrenti, utilizzato dai classici web server.

AWS Lambda

AWS Lambda è un servizio di elaborazione serverless che esegue codice in risposta a determinati eventi e gestisce automaticamente le risorse di elaborazione in uso. Può essere usato per estendere altri servizi AWS con logica personalizzata oppure creare servizi di back-end.

Serverless Framework

Serverless Framework è un web framework gratuito e open-source scritto tramite `Node.js`. Serverless è un framework per creare applicazioni esclusivamente su AWS Lambda. Un'applicazione Serverless può essere composta da poche `lambda functions` per portare a termine semplici tasks o da molte `lambda functions` per creare ad esempio un intero back-end.

DynamoDB

Amazon DynamoDB è un servizio database NoSQL, il quale permette di immagazzinare documenti e grafici tra i suoi dati. E' veloce e flessibile, ideato per tutte le applicazioni che richiedono una latenza costante non superiore a una decina di millisecondi, e grazie alle sue caratteristiche si presta perfettamente come supporto per applicazioni Web.

API.AI

API.AI è una piattaforma di conversazione che permette interazioni sofisticate con il linguaggio naturale. Le applicazioni sviluppate su questa piattaforma sono costituite da Agent, i quali si occupano di trasformare il linguaggio naturale in dati processabili. Tali Agent sono a loro volta costituiti da Intent, che hanno il compito di associare la richiesta dell'utente ad una determinata azione del software, ed Entity, che sono strumenti per estrarre dal linguaggio naturale i parametri attesi.

JWT

JWT è uno standard basato su JSON per creare token di accesso che possono sostenere richieste. I token sono firmati dalla chiave del server, quindi il client può verificare se un token è legittimo. I token sono pensati per essere compatti, senza contenere caratteri invalidi per gli URL e usabili principalmente nei contesti Single sign-on (SSO). Le richieste JWT possono essere usate solitamente per passare l'identità di un utente autenticato tra un identity provider e un service provider. I token inoltre possono essere autenticati e criptati.

Web Speech API

Web Speech API fornisce due aree distinte di funzionalità, il riconoscimento vocale e la sintesi vocale (conosciuta anche come text to speech o tts). Il riconoscimento vocale è acceduto attraverso l'interfaccia SpeechRecognition, la quale fornisce la possibilità di riconoscere il contesto vocale da un input vocale e risponde appropriatamente. La sintesi vocale è acceduta tramite l'interfaccia SpeechSynthesis, una componente text-to-speech che permette ai programmi di leggere testo.

Speaker recognition

Speaker Recognition API è un servizio Microsoft che identifica utenti individuali, e li autentica per mezzo della voce. Al suo interno utilizza JSON per lo scambio dei dati e API Keys per l'autenticazione.

STT IBM Watson

Lo Speech to Text Watson converte voce in testo scritto. Per trascrivere la voce umana accuratamente il servizio utilizza il machine learning per combinare informazioni riguardo la grammatica e la struttura del linguaggio con la composizione del segnale audio. Il servizio fornisce trascrizioni sempre migliori in base a quanto dialogo viene ascoltato.

Request promise

E' un modello che implementa l'esistenza delle Promises come risposta a una serie di richieste che non possono essere soddisfatte immediatamente. Una promessa è un risultato che verrà reso

disponibile non appena possibile se è possibile mantenere la promessa (fulfilled) oppure è un errore se non la si può mantenere (rejected).

AWS Lambda

AWS Lambda è un servizio di cloud-computing fornito da Amazon, il quale permette l'implementazione di una architettura serverless.

Esso permette l'esecuzione di codice senza preoccuparsi della gestione di server o di tutte le risorse necessarie all'esecuzione del codice, in termini di tempo, spazio e computabilità.

AWS Lambda permette di eseguire codice su una piattaforma ad alta affidabilità, a patto che il codice sia scritto in un linguaggio supportato. Nel nostro caso, faremo utilizzo di Node.js 4.3.2, supportato da AWS Lambda.

Inoltre, questo servizio può essere utilizzato per eseguire del codice in risposta ad eventi.

La signature delle Lambda Function è la seguente:

```
function(event, context,) {  
    ...  
}
```

dove:

- **event** è l'oggetto che contiene i dati della richiesta ricevuta dall'API Gateway. In particolare, contiene i seguenti campi:
 - **body**: campo di tipo **String** contenente il corpo della richiesta;
 - **pathParameters**: oggetto contenente i parametri passati all'API Gateway attraverso il path dell'URL;
 - **urlQueryParameters**: oggetto contenente i parametri passati all'API Gateway attraverso query nell'URL.
- **context** è l'oggetto contenente i dati relativi alla richiesta e, in caso di Lambda Proxy Integration, contiene anche i seguenti metodi:
 - **succeed**: metodo da utilizzare per mandare una risposta all'API Gateway in caso di successo. La risposta dev'essere un oggetto contenente i seguenti campi:
 - * **headers**: array associativo nel quale la chiave indica il nome di un header HTTP da mandare nella risposta, il quale valore associato è una stringa contenente il valore di tale header;
 - * **statusCode**: attributo intero contenente il codice HTTP che dovrà avere la risposta;
 - * **body**: stringa contenente il corpo della risposta da mandare.
 - AWS Lambda uses this parameter to provide your handler the runtime information of the Lambda function that is executing. For more information, see The Context Object (Node.js).