



INGEGNERIA DEL SOFTWARE
a.a. 2018/2019

Capitolato C4 - MegAlexa

Manuale Sviluppatore

Componenti:

Sonia MENON
Alberto MIOLA
Andrea PAVIN
Alessandro PEGORARO
Matteo PELLANDA
Pardeep SINGH
Luca STOCCO

Destinatari:

Prof. Tullio VARDANEGA
Prof. Riccardo CARDIN
zero12

Informazioni sul documento

<i>Responsabile</i>	Alberto MIOLA
<i>Verifica</i>	Alessandro PEGORARO, Pardeep SINGH
<i>Redazione</i>	Matteo PELLANDA, Sonia MENON Luca STOCCO, Andrea PAVIN
<i>Uso</i>	Esterno
<i>Stato</i>	Approvato
<i>Email</i>	duckware.swe@gmail.com
<i>Riferimento</i>	Capitolato C4 - MegAlexa

Descrizione

Documento esterno, disponibile per la visione alla proponente *Zero12*, che delinea le funzionalità tecnologiche e il codice realizzato prodotto dal Gruppo *duckware*.

Versione 0.1.0 del
06 Aprile 2019

Indice

Registro delle modifiche

1	Introduzione	1
1.1	Scopo del documento	1
1.2	Scopo del prodotto	1
1.3	Riferimenti	1
2	Installazione	2
2.1	Requisiti software	2
2.2	Requisiti per Windows	2
2.3	Requisiti per Mac OSX	2
2.4	Installazione	2
2.5	Esecuzione	3
2.5.1	Applicazione Android	3
2.5.2	Skill Alexa	3
3	Preparazione ambiente di lavoro	4
3.1	Scopo del paragrafo	4
3.2	Node.js	4
3.3	Android Studio	4
3.4	Amplify	4
3.5	Java Development Kit	5
4	Test	6
4.1	Scopo del paragrafo	6
4.2	Test su Android Studio	6
4.3	Test su AWS	6
5	Architettura applicazione Android	7
5.1	Amplify	12
5.2	Generazione API	13
6	Architettura Skill	15
6.1	Back-end	15
6.1.1	Handler	16
6.1.2	Action	17

6.1.3	ActionFactory	17
6.1.4	Interazione con il database	18
7	Licenza	19

Elenco delle tabelle

1	Registro delle modifiche
---	------------------------------------

Registro delle modifiche

Ver.	Data	Autore	Ruolo	Descrizione
0.0.9	2019-04-06	Alberto MIOLA	Responsabile	Approvazione per rilascio del documento in RQ
0.0.9	2019-04-06	Pardeep SINGH	Verificatore	Superamento verifica documento
0.0.8	2019-04-01	Matteo PELLANDA	Amministratore	Correzione errori ortografici in §3
0.0.7	2019-04-01	Alberto MIOLA	Amministratore	Inserimento §4
0.0.6	2019-04-01	Matteo PELLANDA	Amministratore	Correzione errori ortografici e di contenuto in §2
0.0.5	2019-04-01	Alberto MIOLA	Amministratore	Inserimento §3
0.0.4	2019-03-31	Alberto MIOLA	Amministratore	Inserimento §2
0.0.3	2019-03-30	Matteo PELLANDA	Amministratore	Correzione errori ortografici e numerazione sottocapitoli in §1
0.0.2	2019-03-29	Alberto MIOLA	Amministratore	Inserimento §1
0.0.1	2019-03-21	Matteo PELLANDA	Amministratore	Creazione scheletro del documento

Tabella 1: Registro delle modifiche

1 Introduzione

1.1 Scopo del documento

Lo scopo di questo documento è quello di fornire tutte le informazioni necessarie per estendere, migliorare e correggere *SwetlApp*. Ci saranno ulteriori informazioni riguardanti la configurazione dell'ambiente di sviluppo che consentiranno di lavorare nelle stesse condizioni del team *duckware*. Questa guida è stata scritta tenendo in considerazione i sistemi operativi Microsoft Windows, MacOS Mojave e sistemi UNIX. Nel caso venissero utilizzati altri sistemi operativi, potrebbero esserci dei problemi di compatibilità ma si prevede che le differenze siano minime o nulle.

1.2 Scopo del prodotto

Lo scopo di questo prodotto è quello di creare un'applicazione, ovvero una *Skill*, per l'assistente vocale *Amazon Alexa* in grado di eseguire dei *workflow* personalizzati dall'utente per mezzo di un'applicazione Android. Il front-end del sistema consiste di in un'applicazione nativa creata in Java. Il back-end è stato implementato con i servizi di Amazon Web Services: Amplify, Cognito e AppSync; le skill di Alexa sono state realizzate con NodeJS.

1.3 Riferimenti

- **Git**
<https://git-scm.com/>
- **Node**
<https://nodejs.org/en/>
- **Java**
<https://www.java.com/en/>
- **Android**
<https://developer.android.com/>
- **Amplify**
<https://aws.amazon.com/it/amplify/>
- **AppSync**
<https://aws.amazon.com/it/appsync/>
- **GraphQL**
<https://graphql.org/>

2 Installazione

2.1 Requisiti software

- Java Development Kit (jdk) 1.8.151 o superiore;
- Android Studio 3.3 o superiore;
- NodeJS 10.15 o superiore;
- Mozilla Firefox v.53 o superiore;
- AWS Amplify.

Note. La console di *Amplify* può essere integrata in *Android Studio* e consente di generare tutti gli SDK necessari senza accedere alla console di AWS. Il team duckware ha utilizzato questo approccio perché automatizza l'import ed il setup delle librerie all'interno dell'applicazione.

Tuttavia è anche possibile utilizzare la versione standalone della console ma in questo caso è richiesto il setup manuale delle librerie e l'importazione dei file necessari.

Di seguito vengono riportati i requisiti per l'installazione del software sopra elencato per le distribuzioni di Windows e Mac OSX:

2.2 Requisiti per Windows

- Sistema operativo: Windows 10, 32 o 64 bit;
- RAM: 8GB di RAM;
- Disco fisso: 4GB di spazio libero richiesto;
- Connessione ad internet richiesta.

2.3 Requisiti per Mac OSX

- Sistema operativo: MacOS 10.14 Mojave o superiore, 64 bit;
- RAM: 8GB di RAM;
- Disco fisso: 4GB di spazio libero richiesto;
- Connessione ad internet richiesta.

2.4 Installazione

Creare una cartella per il progetto *SwetlApp* ed utilizzando Git da command line fare una copia del seguente comando:

```
git clone https://github.com/Andreapava/swetlAPP.git
```

2 Installazione

altrimenti fare un clone del progetto da client GUI di Git copiando l'indirizzo HTTPS <https://github.com/Andreapava/swetlAPP.git>.

Creare poi una cartella per il codice sorgente delle Skill per Alexa e, utilizzando nuovamente Git con CLI oppure il client GUI, utilizzando il seguente comando:

```
git clone https://github.com/duckware-swe/Swetlapp.git
```

altrimenti fare il clone del progetto copiando l'indirizzo HTTPS <https://github.com/duckware-swe/Swetlapp.git>

2.5 Esecuzione

2.5.1 Applicazione Android

Per poter utilizzare e testare *SwetlApp* è sufficiente eseguire il comando *Build and Run* da Android Studio e fare il deploy dell'applicazione su un dispositivo reale o emulato. Nel secondo caso, sono sufficienti le impostazioni di default generate dall'AVD manager.

2.5.2 Skill Alexa

Per eseguire la Skill per Alexa, è necessario seguire questi passi per poter eseguire il codice su AWS:

1. Accedere a <https://developer.amazon.com/it/alexa-skills-kit/> ed effettuare il login;
2. Navigare su *Le mie console Alexa* e cliccare su *Skills*;
3. Utilizzare i bottoni proposti dall'interfaccia per creare, modificare e cancellare skill.

3 Preparazione ambiente di lavoro

3.1 Scopo del paragrafo

Questo paragrafo spiega come configurare l'ambiente di lavoro di modo che possa essere replicato da chiunque per operare nelle stesse condizioni del team *duckware*. Tutti i tool che verranno elencati non sono obbligatori ma evitano l'insorgere di problemi legati alla compatibilità.

3.2 Node.js

Per la creazione della Skill di Alexa è necessario installare Node.js, disponibile sia per Windows che per Mojave. Successivamente, bisognerà aprire la console di Node.js ed eseguire:

```
npm install aws-sdk
```

In questo modo verranno installati tutti i pacchetti necessari per poter lavorare sul codice delle Skill. Si consiglia l'aggiornamento frequente della libreria *aws-sdk* che viene settimanalmente aggiornata.

3.3 Android Studio

L'IDE scelto per lo sviluppo di *SwetlApp* è Android Studio, il tool ufficiale di Google scaricabile gratuitamente e disponibile sia per Microsoft Windows sia per MacOS Mojave.

Dal sito ufficiale è disponibile anche una versione per Linux ma questa non risulta essere stabile in tutte le distribuzioni e si possono riscontrare diversi problemi nella configurazione dei dispositivi virtuali.

3.4 Amplify

Amplify è un framework di AWS che facilita l'integrazione dei servizi di AWS nelle applicazioni Android e iOS. Nel progetto *SwetlApp* è stato utilizzato per aggiungere all'applicazione il servizio di autenticazione Cognito e per generare le API.

L'installazione di Amplify avviene all'interno della CLI di Android Studio eseguendo il seguente comando, premesso che Node.js sia stato installato:

```
npm install -g @aws-amplify/cli
```

Per configurare l'ambiente al primo avvio e scaricare le dipendenze necessarie bisognerà utilizzare sempre nella CLI questo comando:

```
amplify configure
```

Per impostare permessi e ruoli e collegare il progetto al backend su cloud AWS che si aggungerà sarà necessario spostarsi nella root del progetto e dare il comando

3 Preparazione ambiente di lavoro

```
amplify init
```

Per modificare le API (riferite ad un endpoint GraphQL) sarà necessario modificare il file `SwetlApp \amplify \backend \api \SwetlApp \schema.graphql` e dare il comando

```
amplify api push
```

Per aggiungere plugin (per una spiegazione in dettaglio su questi riferirsi alla sezione Architecture del link di Amplify fornito) è necessario dare il comando:

```
amplify <category> add
```

dove `<category>` si riferisce al tipo di plugin.

3.5 Java Development Kit

Con l'installazione di Android Studio viene automaticamente installata la versione più recente del JDK, compatibile con l'IDE. È anche possibile installare la propria versione del kit anche se ci potrebbero essere dei problemi di compatibilità. Su Windows andrà installato *JDK* mentre su MacOS *OpenJDK*.

4 Test

4.1 Scopo del paragrafo

Questo paragrafo ha lo scopo di indicare agli sviluppatore come controllare le azioni del proprio codice e la sua sintassi.

4.2 Test su Android Studio

In Android Studio è possibile eseguire test sia per l'applicazione, con Java e JUnit, sia per la Skill utilizzando Amplify ed il suo framework interno.

- I test per il codice dell'applicazione sono presenti in Android Studio all'interno dell'apposito progetto *test*. Questi sono stati creati con il supporto della libreria *jUnit 5* e possono essere eseguiti direttamente dall'editor di Android Studio cliccando sul bottone verde di *Run*.
- I test per il *GraphQL* sono integrati nella console di Amplify.

4.3 Test su AWS

Per eseguire test sulla Skill e su Alexa è necessario accedere al portale sviluppatori¹ di AWS ed entrare nell'area dedicata alle Skill Alexa. In quest'area si potranno eseguire dei test preimpostati e vedere i risultati su:

- **Logger:** una finestra che mostra un resoconto di tutte le azioni eseguite dal backend e le risposte ricevute da Alexa:
- **Alexa:** se un dispositivo fisico Alexa è disponibile, verranno eseguiti direttamente i test su di esso.

¹<https://developer.amazon.com/alexa-skills-kit>

5 Architettura applicazione Android

Per avere un overview generale dell'applicazione è utile osservare il seguente diagramma dei package:

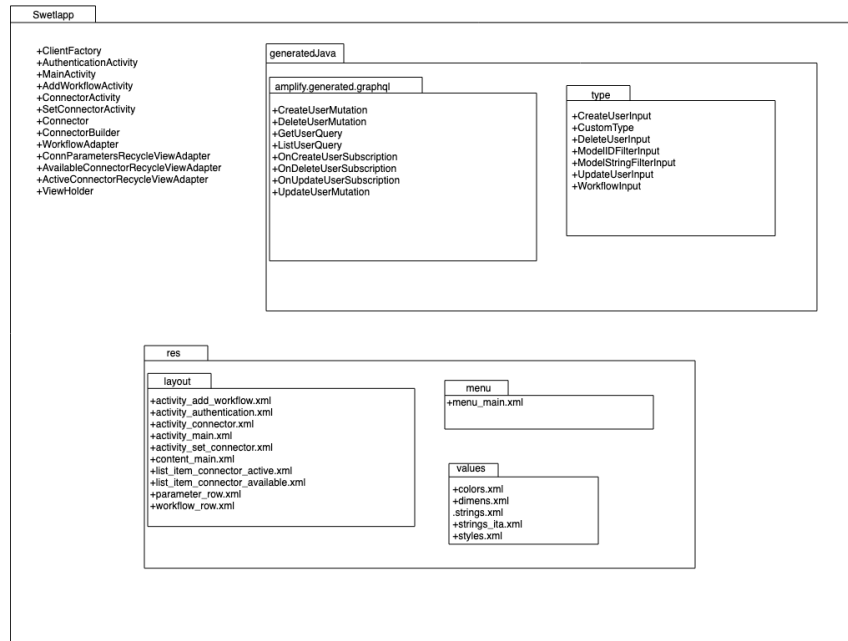


Figura 1: Package diagram SwetlApp

L'applicazione contiene le seguenti activity:

- *AuthenticationActivity*: è la prima activity che viene aperta dall'app, permette la registrazione, il login e verifica se l'utente è loggato o meno. Il form di inserimento dei dati e la loro gestione è generato da AWS Cognito;

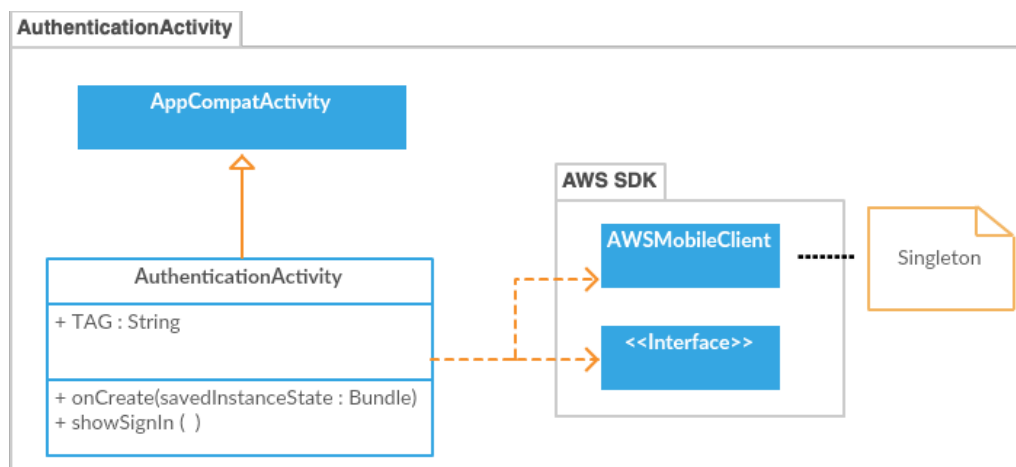


Figura 2: Diagramma della classe AutenticationActivity

5 Architettura applicazione Android

- *MainActivity*: è la schemata principale che mostra la lista di Workflow dell'utente;

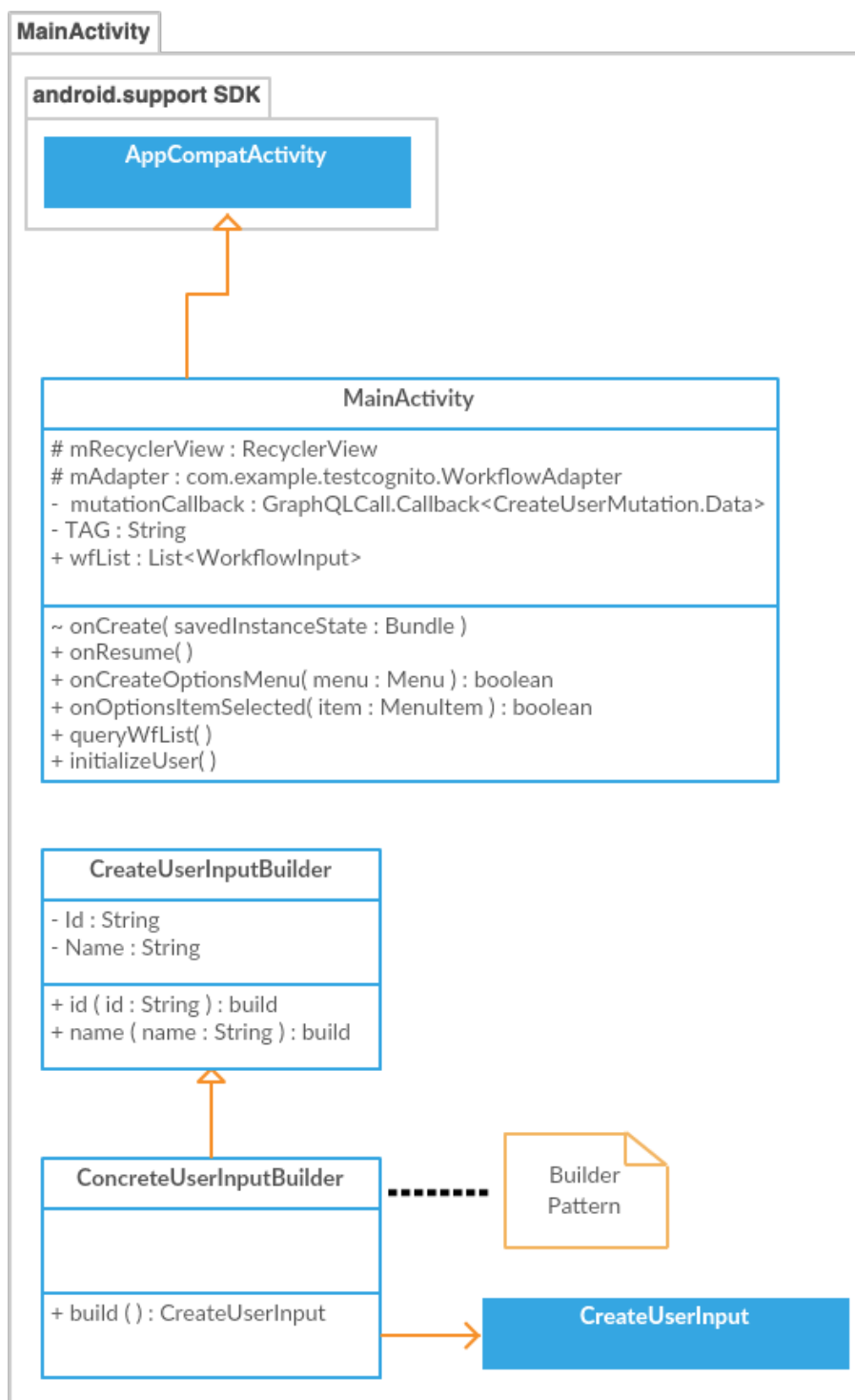


Figura 3: Diagramma della classe MainActivity

- *AddWorkflowActivity*: è l'activity che consente di inserire un nuovo workflow;

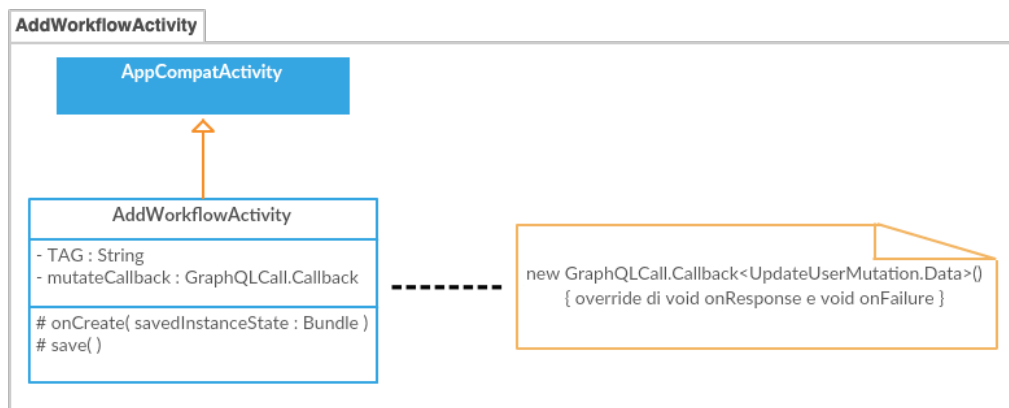


Figura 4: Diagramma della classe AddWorkflowActivity

- *ConnectorActivity*: è l'activity che consente di gestire i connettori relativi al workflow su cui si sta operando;

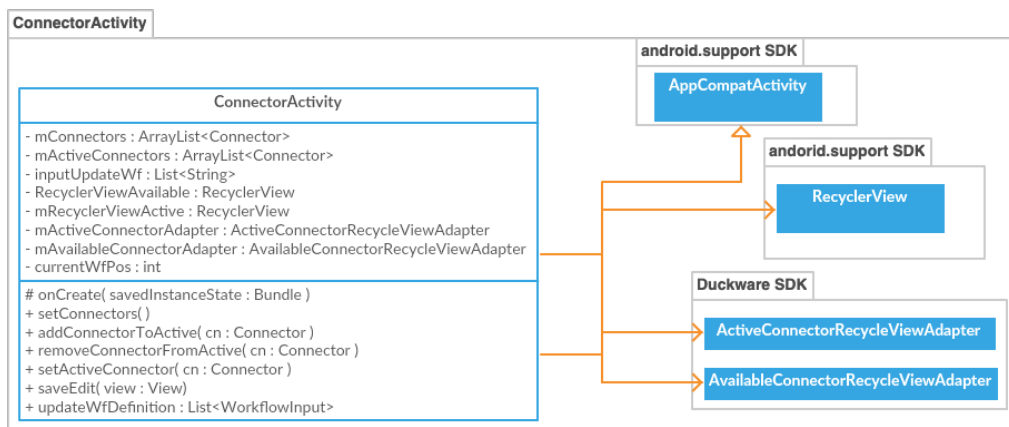


Figura 5: Diagramma della classe ConnectorActivity

- *SetConnectorActivity*: è l'activity che consente di gestire i parametri relativi al connettore su cui si sta operando;

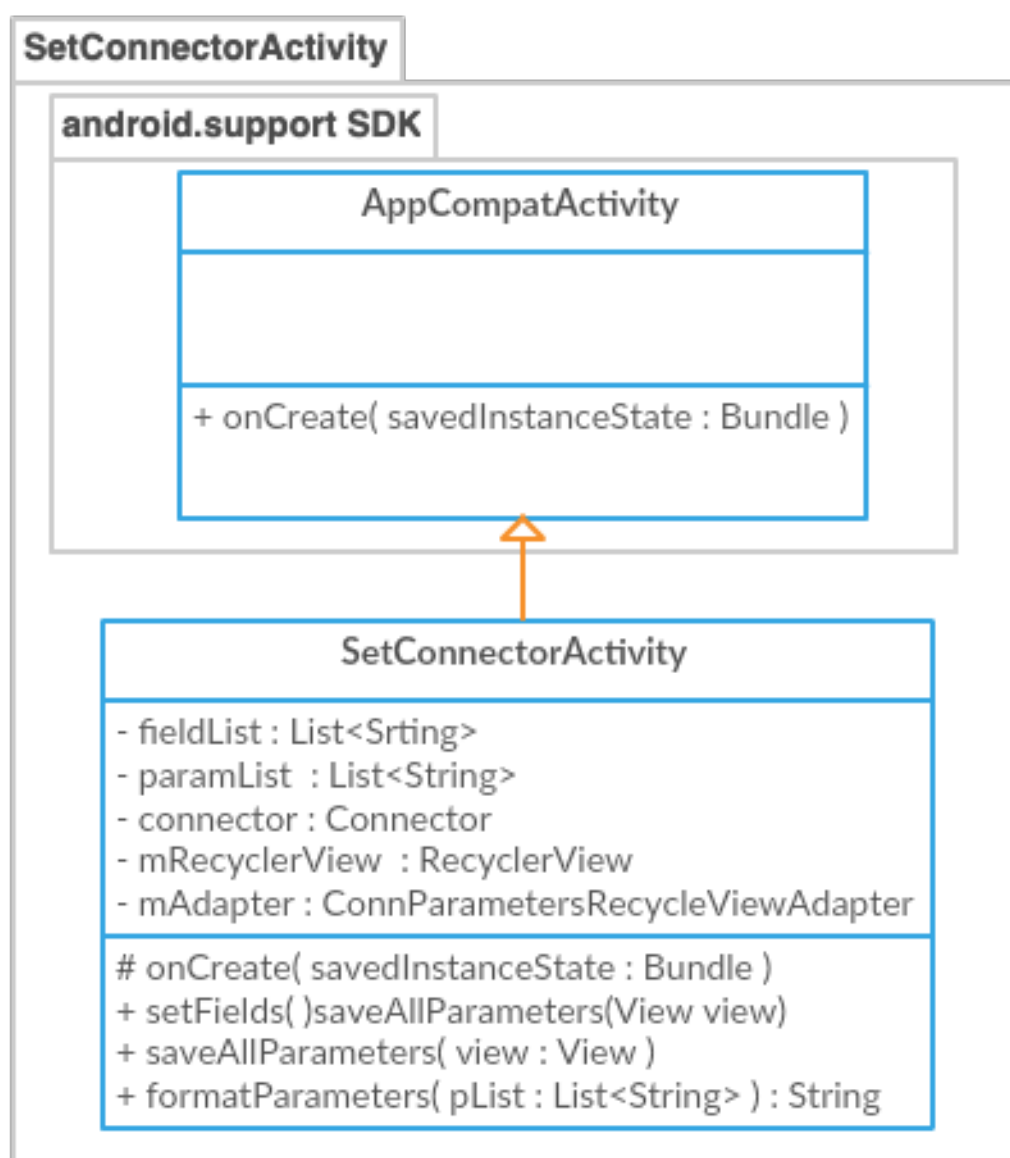


Figura 6: Diagramma della classe SetConnectorActivity

È inoltre presente la classe di supporto `Connector.java` non generata automaticamente da Amplify:

5 Architettura applicazione Android

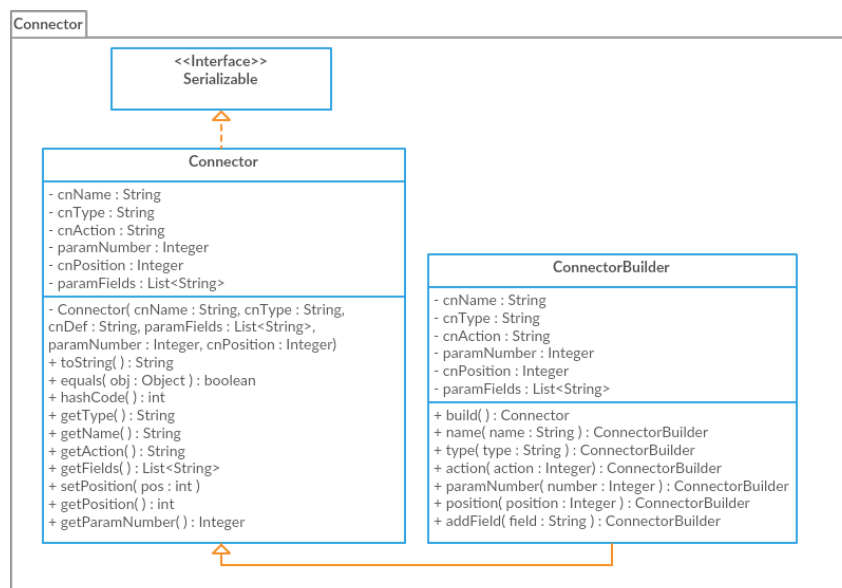


Figura 7: Diagramma della classe Connector

Questa utilizza un builder pattern poichè i connettori vengono istanziati con attributi eterogenei e a volte vuoti.

Ogni Activity potrà comunicare con le API solamente utilizzando il client generato dalla classe *ClientFactory*.

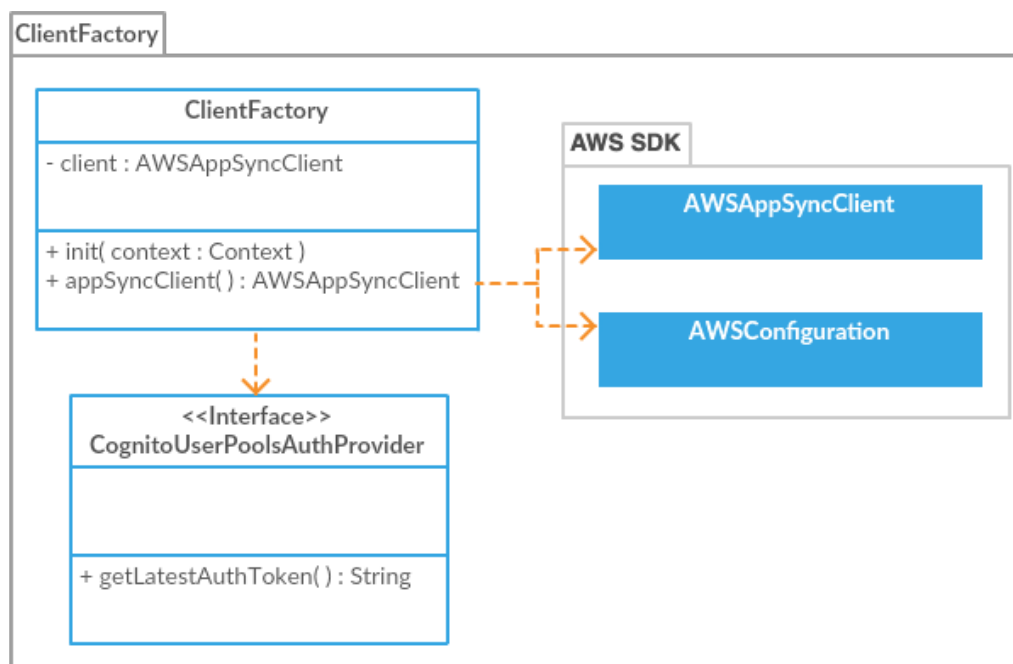


Figura 8: Classe ClientFactory

5 Architettura applicazione Android

Le risorse dell'interfaccia grafica sono definite nel namespace `res` e comprendono i file in linguaggio xml che descrivono gli elementi delle view (layout) e i valori importati da questi elementi (values) quali:

- colori;
- dimensioni;
- stili;
- stringhe per lingua italiana;
- stringhe per lingua inglese;

Oltre alle activity e alle risorse xml, è importante notare che tutto il codice necessario alle operazioni sulle risorse backend è definito nel namespace `generatedJava`, tutto il codice contenuto in questo namespace è generato in automatico da Amplify.

5.1 Amplify

Amplify è il framework di Amazon utilizzato per generare un SDK che permette la comunicazione dell'applicazione Android con i servizi in cloud di AWS. La CLI di Amplify consente di sincronizzare i servizi online con l'applicazione, generare librerie di supporto e fornire strumenti di test allo sviluppatore.

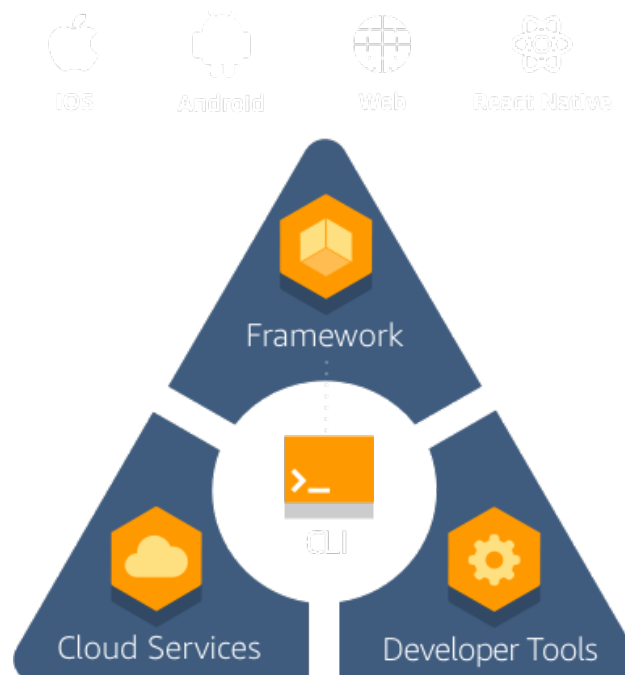


Figura 9: AWS Amplify

L'integrazione di Amplify in Android Studio consente di delegare controllo e creazione del backend dell'applicazione, noi abbiamo utilizzato Amplify principalmente per due funzioni:

5 Architettura applicazione Android

- aggiunta di AWS Cognito.
- aggiunta di API GraphQL.

I metodi per interfacciarsi con Cognito vengono forniti da una libreria di Amazon mentre i metodi per fare query e mutazioni all'endpoint GraphQL vengono generati da Amplify.

Questo è un esempio di query per inserire dati relativi ad un utente:

```
CreateUserInput input = CreateUserInput.builder()
    .id(AWSMobileClient.getInstance().getUsername())
    .name(AWSMobileClient.getInstance().getTokens()
        .getIdToken().getClaim("nickname"))
    .build();

CreateUserMutation addUserMutation = CreateUserMutation.builder()
    .input(input)
    .build();
com.example.swetlapp.ClientFactory.appSyncClient()
    .mutate(addUserMutation).enqueue(callback);
```

CreateUserInput e CreateUserMutation sono due classi generate da Amplify, entrambe sono costruibili tramite builder pattern, AWSMobileClient (singleton) ci permette di comunicare con la user pool di AWS Cognito e ci ritorna informazioni relative all'utente. Quindi si crea l'input per la mutazione, si crea la mutazione stessa, infine esegue questa utilizzando come oggetto di invocazione l'istanza di ClientFactory ritornata dal metodo appSyncClient().

Lo schema generale per "comunicare" con le risorse è sempre questo (a meno di query di sola lettura in cui non è necessario costruire un input).

5.2 Generazione API

Le API nel nostro caso corrispondono a una soluzione per richieste HTTP ad un endpoint GraphQL. Per creare e accedere alle API è necessario il servizio AWS AppSync, anche questo sarà settato da amplify. Componenti principali di AppSync sono:

- un proxy GraphQL che processa le richieste e le mappa su funzioni e trigger;
- operazioni: query e mutazioni;
- data source: nel nostro caso DynamoDB;
- resolver: converte operazioni GraphQL per essere eseguite sul data source effettivo;
- AppSync client: dove sono definite le operazioni GraphQL;

L'ordine degli eventi per l'impostazione del backend è questo:

- settaggio di un endpoint GraphQL tramite CLI Amplify;

5 Architettura applicazione Android

- scrittura dello schema delle risorse in linguaggio GraphQL Schema;
- pubblicazione dello schema;
- generazione delle API su AppSync (automatico)
- generazione del database DynamoDB (automatico)
- generazione dell'SDK per effettuare operazioni definite dallo schema GraphQL (automatico)

Lo schema GraphQL che abbiamo scritto è il seguente:

```
type User @model {
  id: ID!
  name: String!
  workflow: [Workflow!]
}

type Workflow {
  idwf: ID!
  name: String
  def: String
}
```

A partire da questo, in dynamoDB è stata generata una tabella User con i campi:

- id;
- name;
- lista di workflow;

Amplify genera solo le classi java dei tipi con direttiva *@model*, quindi abbiamo a disposizione solo operazioni con il tipo User, per accedere ai workflow, ai connettori dei workflow e ai parametri dei connettori è necessario operare con il file Json definito dalla stringa *def*, per operare agevolmente con i file Json in ambiente Android abbiamo importato le librerie *import org.json.JSONException* e *import org.json.JSONObject*.

Esempio: per generare il Json nella forma:

```
{"action\":\"connector_action\",\"params\": [\"first_param\",\"second_param\"]}
```

il codice Java sarà:

```
Map<String, Object> connMap = new HashMap<>();
connMap.put("action",connector.getAction());
connMap.put("params",pList);
JSONObject jsonObject = new JSONObject(connMap);
```

6 Architettura Skill

L'architettura della skill che risiede in cloud in AWS Lambda è rappresentata dal seguente diagramma:

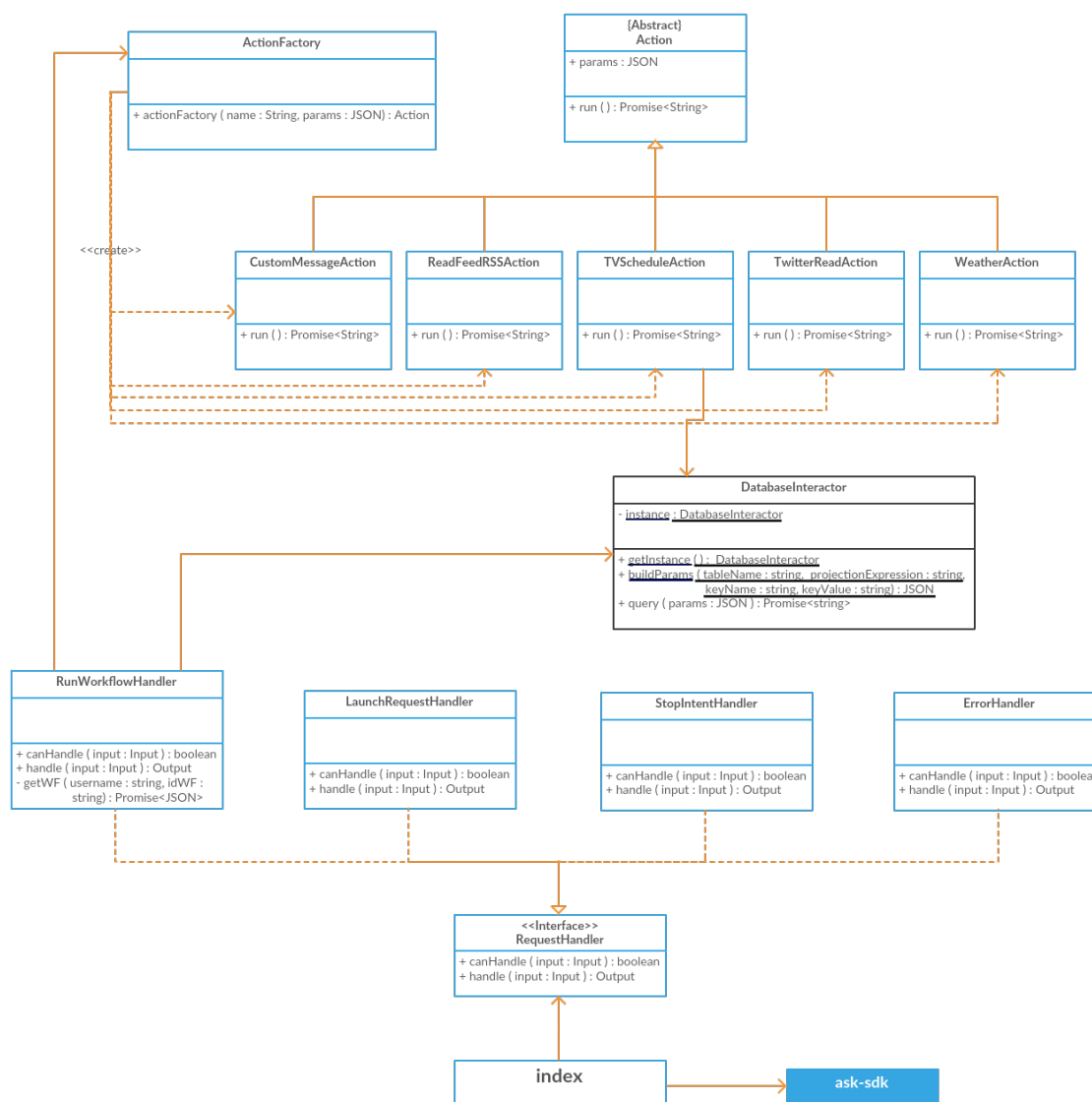


Figura 10: Overview architetturale della skill.

6.1 Back-end

La logica della skill è gestita dal file `index.js` al quale Alexa si appoggerà per l'invio delle richieste derivanti dall'interazione con l'utente. Ogni richiesta inviata incapsula una serie di dati, quali il tipo di intent che la attiva e i parametri forniti dall'utente, denominati slot, e sarà gestita da uno degli handler presenti nel file `index.js`. Per ognuno di questi verrà eseguito il metodo `canHandle()` e, in caso di ritorno positivo, sarà lanciato il relativo metodo `handle()`, interrompendo la ricerca dell'handler adeguato. Nel caso in cui non

venga trovato alcun gestore appropriato, la skill ha un comportamento non definito che porta al suo arresto improvviso; pertanto la best practice prevede l'implementazione di un handler di default che viene eseguito all'occorrenza di errori.

6.1.1 Handler

L'interfaccia `RequestHandler` è fornita dalle librerie di AWS incluse nel package. Espone un metodo `canHandle(input : Input)` che ritornerà un booleano: `true` nel caso in cui l'handler che lo ridefinisce dovrà gestire la richiesta passatagli; `false` altrimenti. Nel caso di ritorno positivo, verrà eseguito il metodo `handle(input : Input)` fornito dall'interfaccia; la sua ridefinizione dovrà elaborare la richiesta in input per poter fornire un ritorno `Output` comprensibile da Alexa. Gli handler sono strutturati con uno strategy pattern descritto nel seguente diagramma:

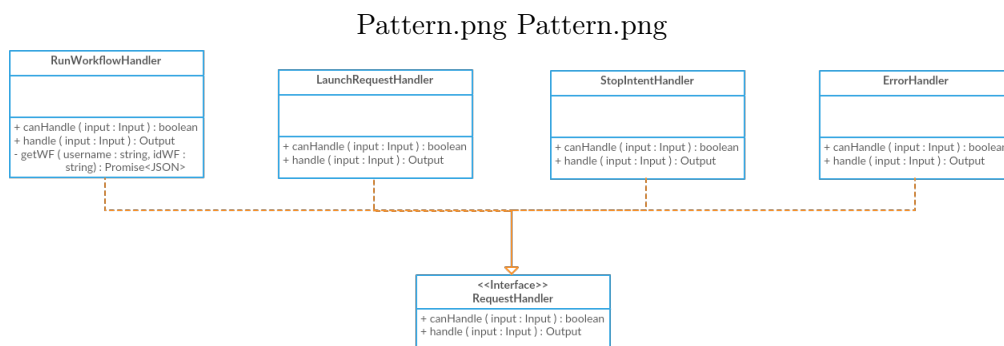


Figura 11: Diagramma della classe Handler

I principali handler da noi definiti sono:

- LaunchRequestHandler***: gestisce il lancio iniziale della skill controllando che l'utente sia autenticato; in caso contrario lo invita ad effettuare il login con una notifica vocale e una push nell'applicazione Amazon Alexa dell'utente;
- RunWorkflowHandler***: viene lanciato quando l'utente richiede l'avvio di un workflow; si occupa di fare una query al database per ottenere i dati del workflow richiesto, quando disponibile, e cominciare la sua esecuzione;
- StopIntentHandler***: elabora la richiesta di arresto della skill da parte dell'utente;
- ErrorHandler***: è il gestore di default che verrà eseguito quando nessuno dei precedenti handler è stato avviato, o nel caso in cui si siano verificati errori durante l'esecuzione della skill;

6.1.2 Action

I workflow sono composti di azioni; la loro struttura è definita da Action, questa è una classe astratta ed espone un metodo `run()` che verrà chiamato per l'effettiva esecuzione dell'azione specifica. La sua ridefinizione delinea il comportamento dell'azione concreta.

Le azioni ridefinite nella Skill SwetlApp sono:

- *CustomMessageAction*: consente ad Alexa di leggere un messaggio definito dall'utente;
- *ReadFeedRSSAction*: consente ad Alexa di ricevere e leggere un feed rss impostato dall'utente;
- *TVScheduleAction*: consente ad Alexa di informare l'utente sulla programmazione quotidiana dei suoi canali TV preferiti;
- *TwitterReadAction*: permette ad Alexa di leggere i tweet di un account Twitter definito dall'utente;
- *TwitterWriteAction*: permette ad Alexa di postare un tweet personalizzato nella bacheca dell'utente;

6.1.3 ActionFactory

Si occupa della costruzione di un oggetto concreto di tipo derivante da Action. Nasconde all'esterno il modo in cui viene scelto quale oggetto costruire, permette di estendere e mantenere velocemente e semplicemente il codice, in quanto per aggiungere una nuova Action sarà sufficiente che questa implementi l'interfaccia, e che venga aggiunto un controllo sul factory perchè questa sia usabile all'esterno. Il controllo per capire qual è il tipo di azione corretto è fatto a partire dal nome dell'azione che viene chiesto come parametro, su questo sarà svolto uno switch case che ritorna l'azione corretta.

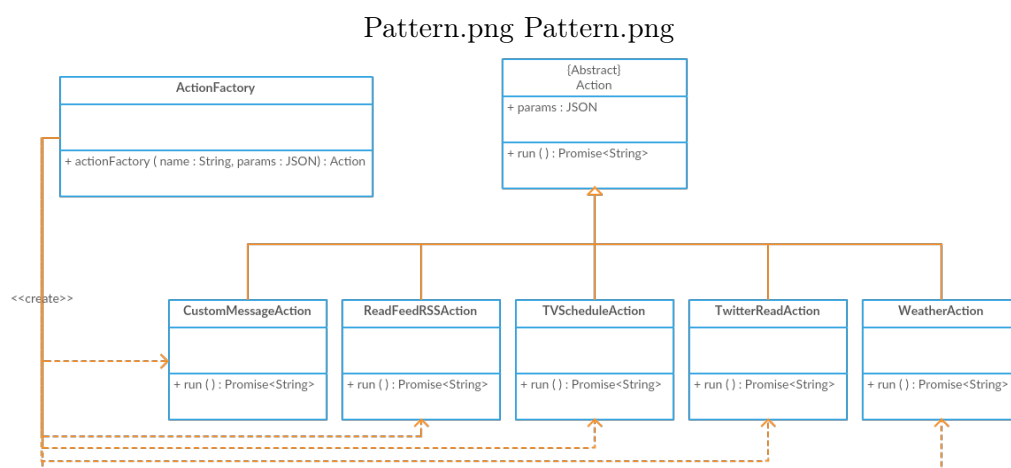


Figura 12: Diagramma della classe Action

6.1.4 Interazione con il database

La connessione al db è gestita con un pattern singleton che quindi impone l'esistenza di un'unica istanza di *DatabaseInteractor* in un dato momento. L'accesso di più client contemporaneamente al database è possibile perchè AWS si occupa poi di gestire le multiple richieste di accesso al db. Fornisce un metodo di supporto *query(params : JSON)* per gestire tutte le interazioni con il db direttamente da DatabaseInteractor.

Pattern.png Pattern.png

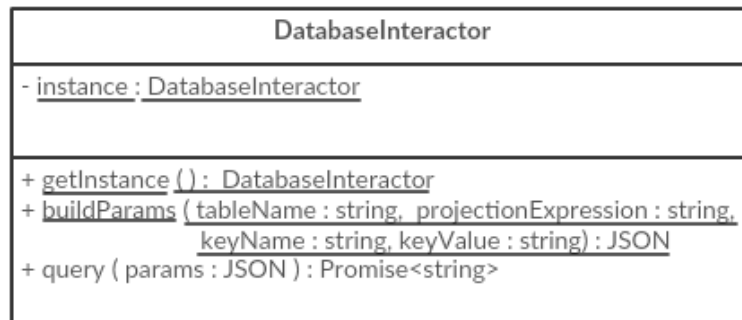


Figura 13: Diagramma della classe DatabaseInteractor

7 Licenza

Copyright 2019 Duckware

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.