



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 1.0.0 del
13 Maggio 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 in Android Studio	6
4.3	Test in ambiente 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	Estensione delle funzionalità	19
7.1	Architettura AWS	19
7.2	Android	19
7.2.1	Estensione delle risorse	19
7.2.2	Accesso a nuove risorse	19
7.2.3	Estensione Front-end	19
7.2.4	Cognito	20
7.2.5	Implementazione nuovi connettori	20
7.3	Skill	20
7.3.1	Implementazione nuovi connettori	20
7.3.2	Connettori che interagiscono con l'utente	20
7.3.3	Collegare il connettore alla Skill	21
8	Licenza	22

Elenco delle tabelle

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

Elenco delle figure

1	Diagramma della classe AuthenticationActivity	7
2	Diagramma della classe MainActivity	8
3	Diagramma della classe AddWorkflowActivity	9
4	Diagramma della classe ConnectorActivity	9
5	Diagramma della classe SetConnectorActivity	10
6	Diagramma della classe Connector	11
7	Classe ClientFactory	11
8	AWS Amplify	12
9	Overview architetturale della skill.	15
10	Diagramma della classe Handler	16
11	Diagramma della classe Action	17
12	Diagramma della classe DatabaseInteractor	18

Registro delle modifiche

Ver.	Data	Autore	Ruolo	Descrizione
1.0.0	2019-05-13	Alberto MIOLA	Responsabile	Approvazione per rilascio del documento in RA
0.2.0	2019-04-11	Alessandro PEGORARO	Verificatore	Superamento verifica documento
0.1.3	2019-05-09	Luca STOCCO	Amministratore	Stesura §7.3
0.1.2	2019-05-07	Alberto MIOLA	Responsabile	Correzione contenuto e forme verbali
0.1.1	2019-05-06	Andrea PAVIN	Amministratore	Stesura §7.2
0.1.0	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	Sonia MENON	Amministratore	Stesura §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	Stesura §3
0.0.4	2019-03-31	Alberto MIOLA	Amministratore	Stesura §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_G Alexa_G* in grado di eseguire dei *workflow_G* personalizzati dall'utente per mezzo di un'applicazione Android. Il *front-end_G* del sistema consiste di in un'applicazione nativa creata in *Java_G*. Il *back-end_G* è stato implementato con i servizi di *AWS_G*: *Amplify_G*, *Cognito_G* e *AppSync_G*; le skill di Alexa sono state realizzate con il linguaggio *Node.js_G*.

1.3 Riferimenti

- **Git**
<https://git-scm.com/>
- **Node.js**
<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;
- Node.js 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 *CLI* fare una copia del seguente comando:

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


2 Installazione

altrimenti clonare il progetto tramite *client_G* GUI di Git, copiando l'indirizzo HTTPS <https://github.com/Andreapava/swetlAPP.git>.

Tramite CLI, eseguire il seguente comando in una nuova cartella, così da poter clonare il codice sorgente della skill per Alexa:

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

In alternativa, clonare il progetto copiando l'indirizzo HTTPS <https://github.com/duckware-swe/Swetlapp.git> nel client 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 *deployment_G* 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 in ambiente 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 pulsanti 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 un utente qualunque, così da poter 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_G* scelto per lo sviluppo di *SwetlApp* è Android Studio, il tool ufficiale di *Google_G* 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_G* di AWS che facilita l'integrazione dei servizi della suite professionale di Amazon, 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_G*.

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 digitare nella CLI questo comando:

```
amplify configure
```

Per impostare permessi e ruoli e collegare il progetto al back-end su cloud AWS che si aggungerà, è necessario spostarsi nella cartella principale del progetto ed eseguire:

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 avviare il comando:

```
amplify api push
```

Per aggiungere *plugin_G* (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 in Android Studio

In Android Studio è possibile eseguire test sia per l'applicazione, con Java e *JUnit_G*, 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 pulsante verde di *Run*.
- I test per il *GraphQL* sono integrati nella console di Amplify.

4.3 Test in ambiente AWS

Per testare la Skill è necessario accedere al portale sviluppatori¹ di Alexa e selezionare la Skill che si vuole testare. 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 Back-end 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/console/ask>

5 Architettura applicazione Android

Tutti i *package_G* sono stati generati automaticamente dall'IDE Android Studio e dal tool di AWS, Amplify. 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 autenticato o meno. Il form di inserimento dei dati e la loro gestione è generato da AWS Cognito;

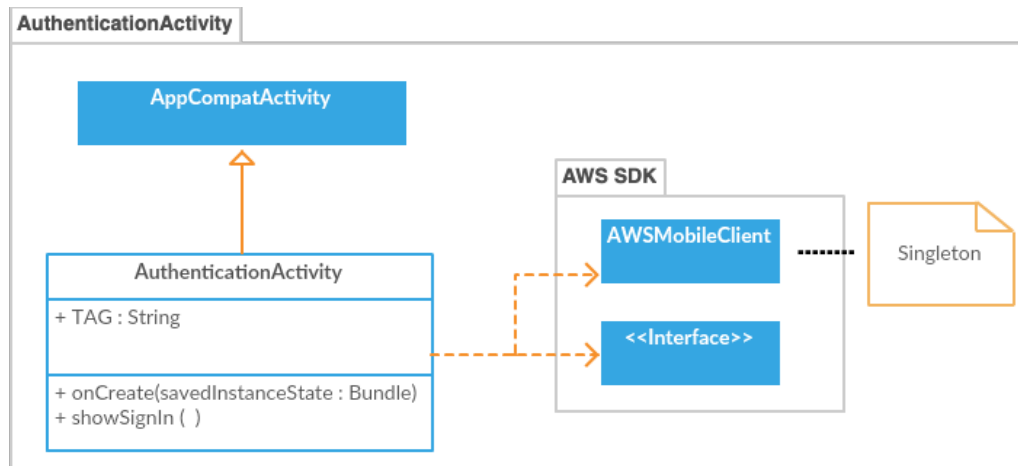


Figura 1: Diagramma della classe AutenticationActivity

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

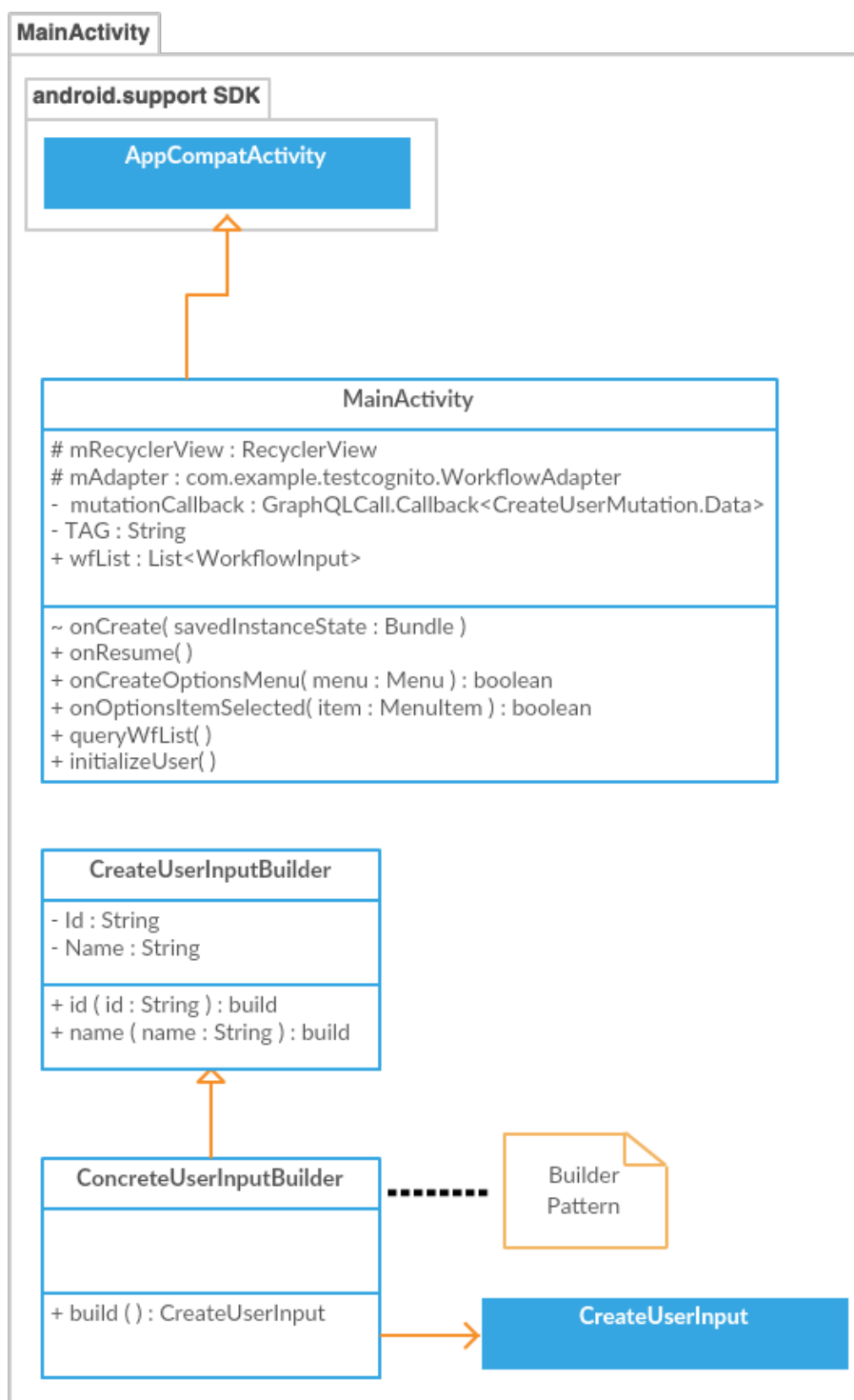


Figura 2: Diagramma della classe MainActivity

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

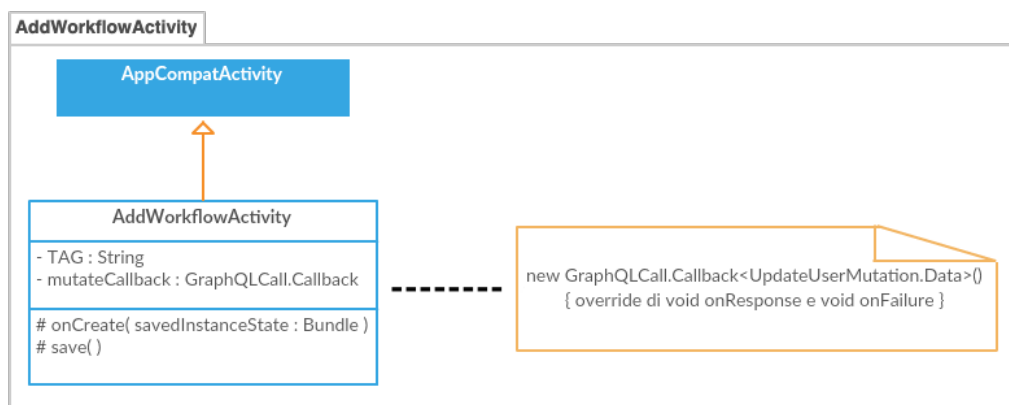


Figura 3: Diagramma della classe AddWorkflowActivity

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

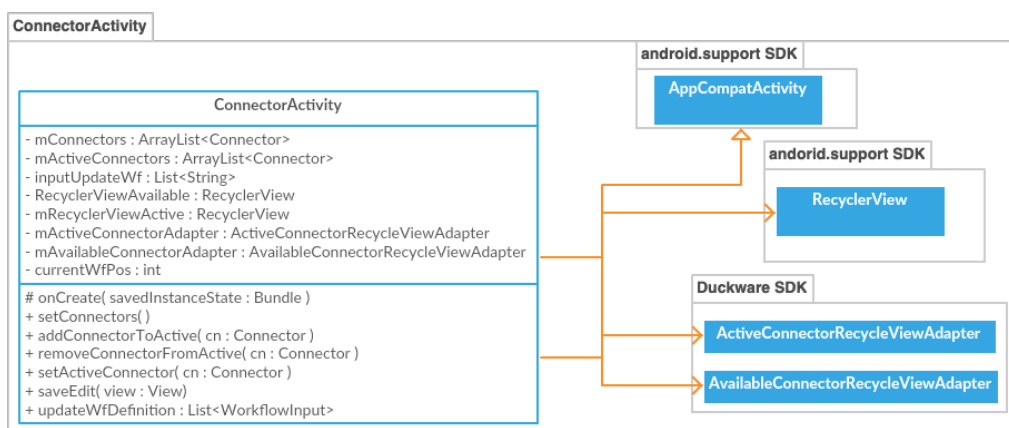


Figura 4: Diagramma della classe ConnectorActivity

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

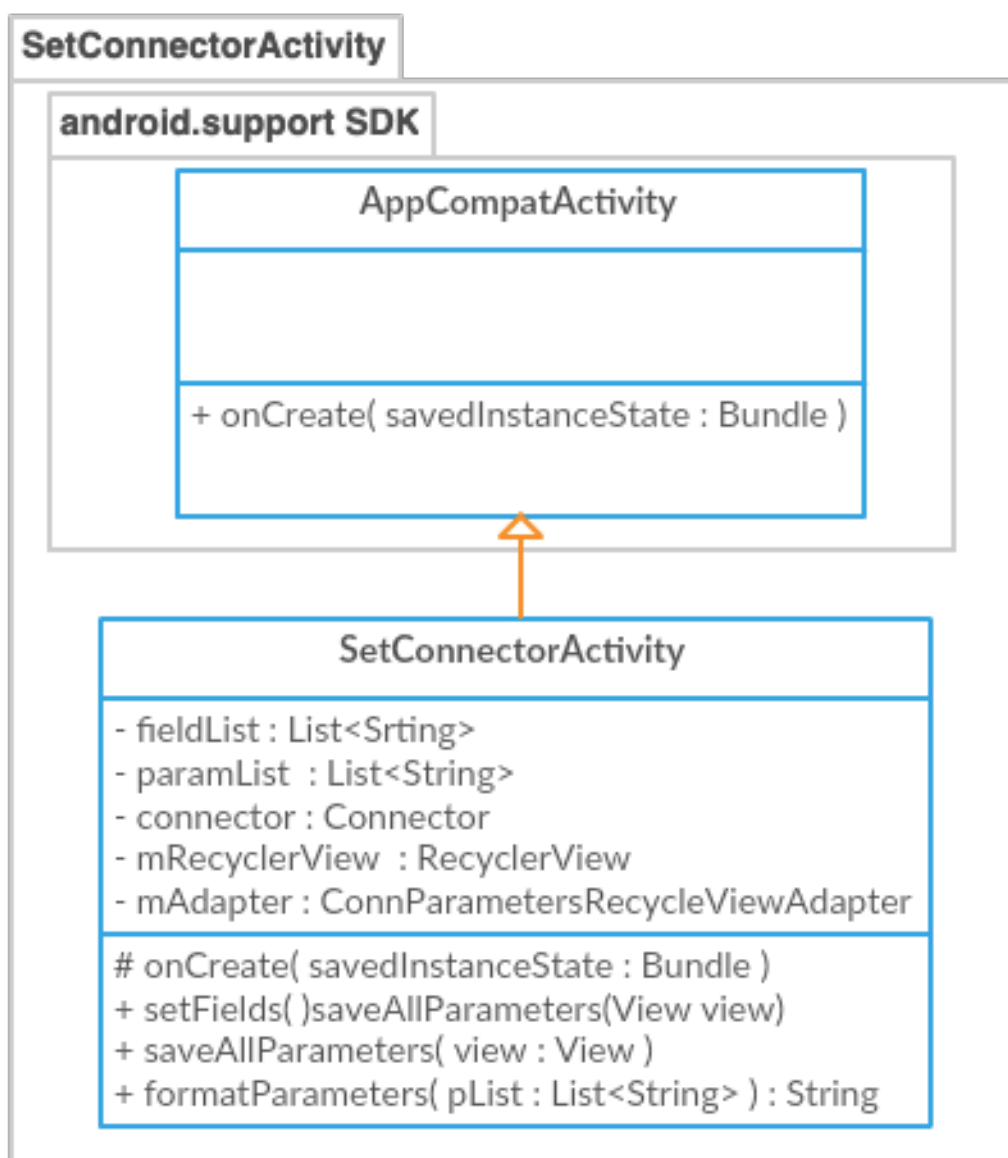


Figura 5: Diagramma della classe SetConnectorActivity

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

5 Architettura applicazione Android

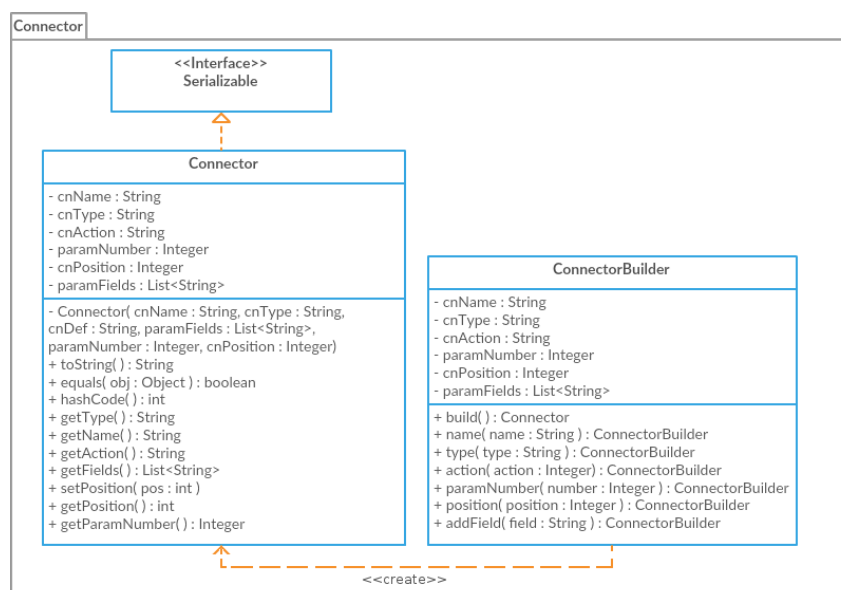


Figura 6: Diagramma della classe Connector

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*.

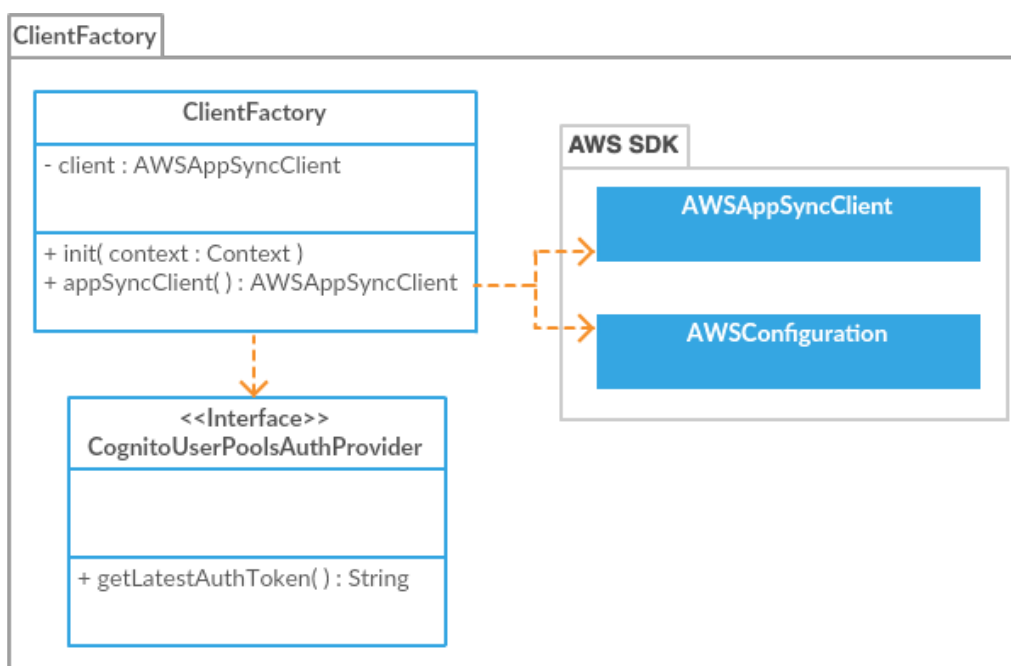


Figura 7: Classe ClientFactory

5 Architettura applicazione Android

Le risorse dell'interfaccia grafica sono definite nel namespace *res* e comprendono i file in linguaggio *XML_G* 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 back-end 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.

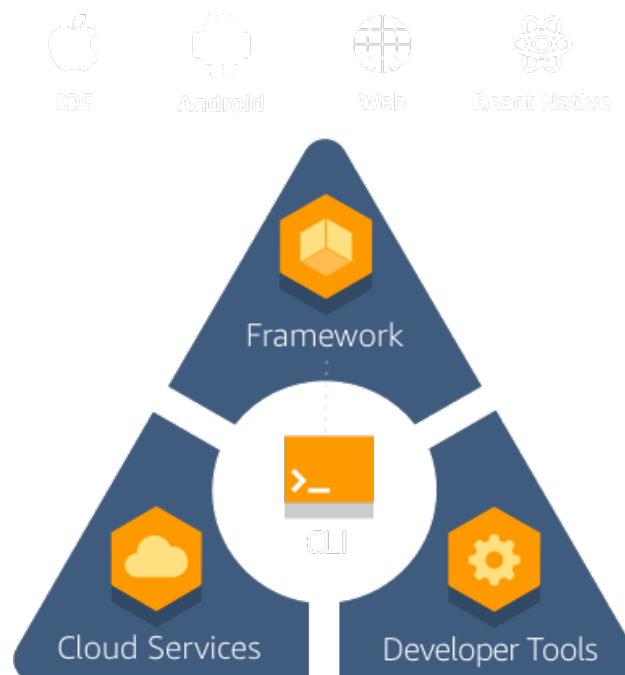


Figura 8: AWS Amplify

L'integrazione di Amplify in Android Studio consente di delegare controllo e creazione del back-end 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 effettuare 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 pattern_G*) permette di comunicare con la user pool di AWS Cognito e ritorna informazioni relative all'utente. Quindi si crea l'input per la mutazione, si crea la mutazione stessa, infine si esegue quest'ultima 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 back-end 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_G* 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 corrispondente 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:

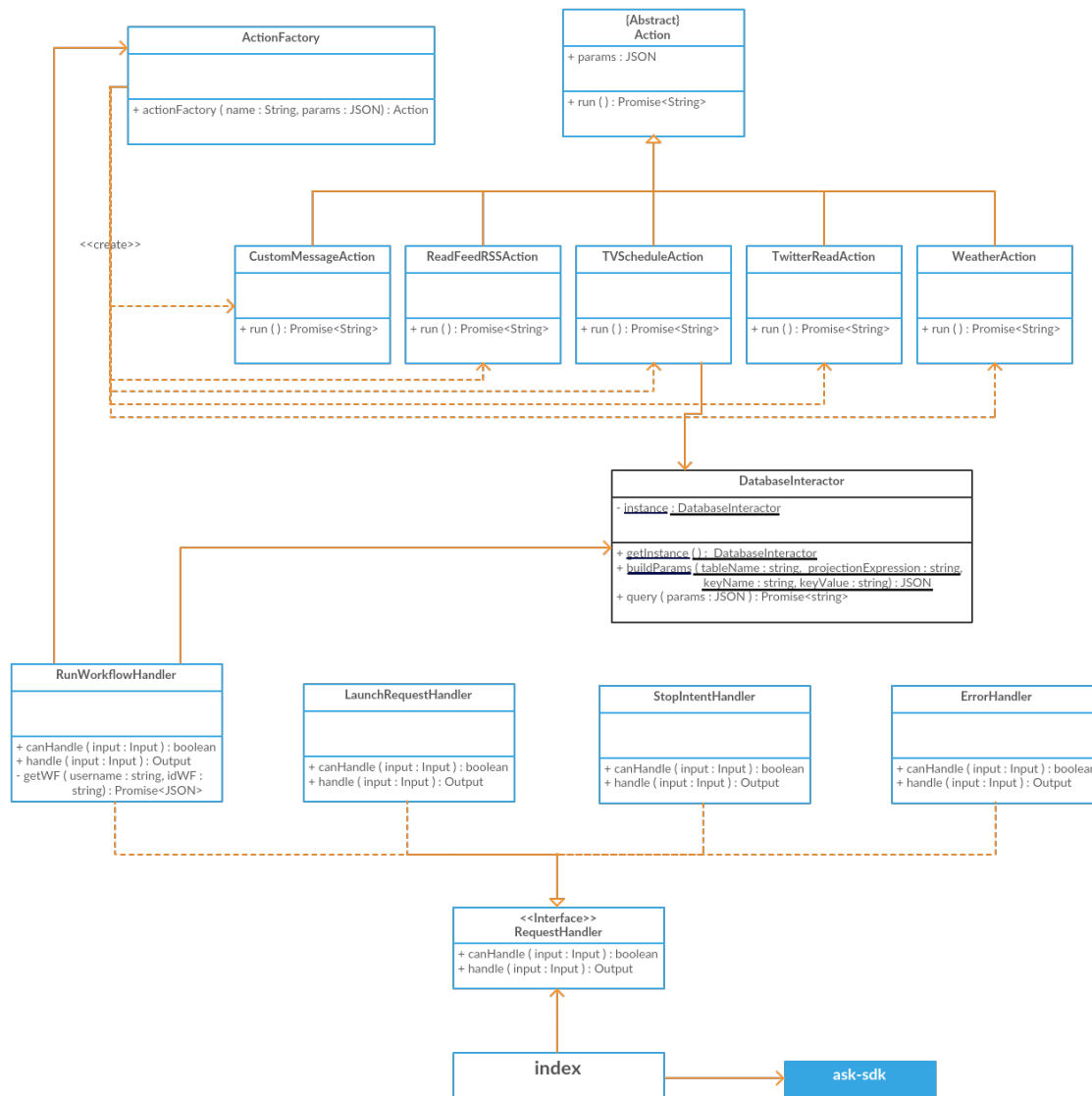


Figura 9: Overview architetturale della skill.

6.1 Back-end

La logica della skill è gestita dal file *index.js* al quale Alexa si appoggerà anche per l'invio delle richieste derivanti dall'interazione con l'utente. Ogni richiesta inviata incapsula una serie di dati, quali il tipo di *intent*_C 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_G* 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:

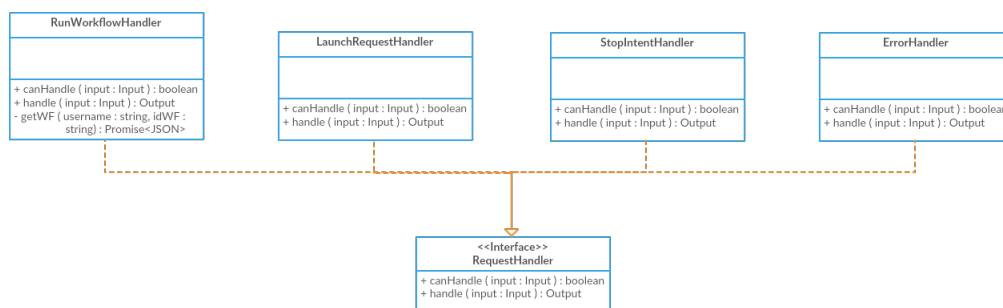


Figura 10: Diagramma della classe Handler

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 effettuare una richiesta 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 da 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 e permette di estendere, 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. Si tratta di un *factory pattern*.

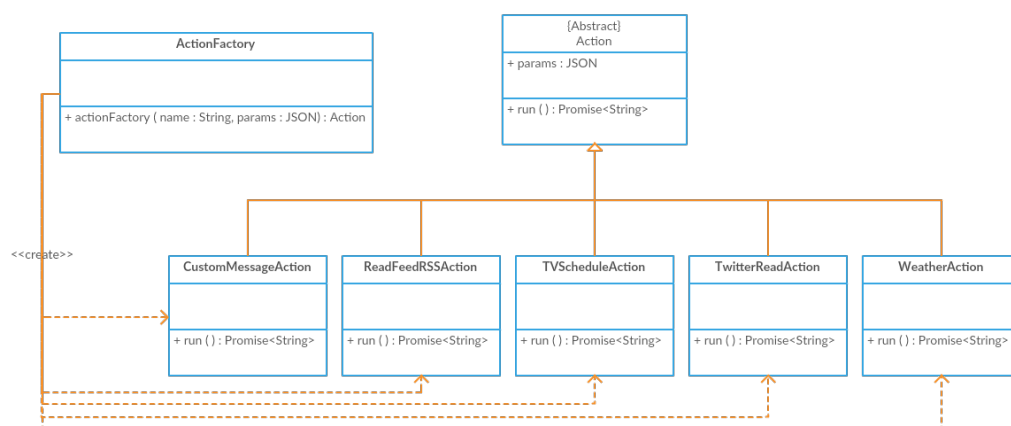


Figura 11: Diagramma della classe Action

6.1.4 Interazione con il database

La connessione al database 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 a DynamoDB. Fornisce un metodo di supporto *query(params : JSON)* per gestire tutte le interazioni con il database direttamente da *DatabaseInteractor*.

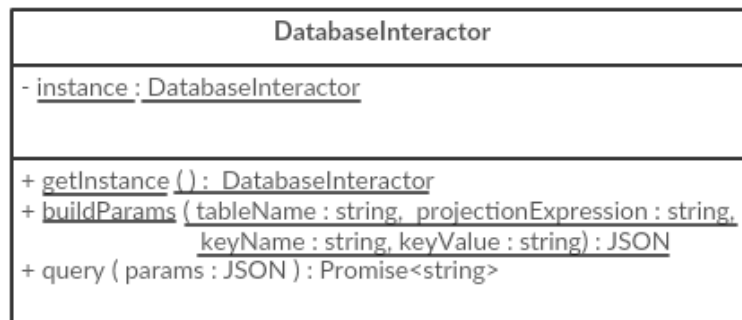


Figura 12: Diagramma della classe DatabaseInteractor

7 Estensione delle funzionalità

Il seguente paragrafo ha lo scopo di aiutare lo sviluppatore che intende apportare modifiche all'architettura su AWS o estendere il codice sorgente.

7.1 Architettura AWS

L'intera architettura viene automaticamente generata dal tool *Amplify* dopo aver interpretato il file GraphQL. Una volta modificato tale file, la console di Amplify genererà il database e l'intera infrastruttura per la comunicazione con l'applicazione Android e la Skill.

7.2 Android

7.2.1 Estensione delle risorse

Per estendere le risorse sarà necessario:

- Localizzare il file `swetlAPP/amplify/backend/api/testcognito/schema.graphql`
- Modificare questo file per adattarlo al modello necessario
- Salvare il nuovo file
- Da CLI sulla cartella root di progetto digitare e inviare **amplify api push**
- Attendere la sincronizzazione con il cloud AWS

7.2.2 Accesso a nuove risorse

Le classi e i metodi di accesso a queste risorse saranno disponibili nella cartella *generated-Java*, il comando **clean** eliminerà eventuali classi associate a risorse non più esistenti.

Le query verranno create con i metodi generati producendo, se necessario, prima un input e poi passandolo come parametro alla query, gli eventi successivi a una query saranno gestiti da una callback con i metodi `onResponse()` e `onFailure()`.

Si consiglia di mettere gli update della view dentro questi due metodi.

7.2.3 Estensione Front-end

Sarà necessario agire sui seguenti file XML:

- **layout** per dichiarare e modificare viste e le posizioni delle stesse
- **strings** per dichiarare e modificare stringhe utilizzate dall'applicazione
- **colors** per dichiarare e modificare i colori utilizzati dalle viste
- **styles** per dichiarare e modificare gli stili utilizzati dalle viste

7.2.4 Cognito

Per modificare le policy e il front-end della drop-in UI di AWS Cognito sarà necessario accedere alla console AWS Cognito da browser e navigare tra le categorie, nella categoria *policy* sarà possibile modificare la sicurezza necessaria per la password e l'obbligatorietà di vari attributi, nella categoria interfaccia si potranno modificare gli elementi della vista della drop-in UI tramite direttive CSS.

7.2.5 Implementazione nuovi connettori

Se si vuole aggiungere un connettore i cui parametri necessitano all'utente solo di digitare in campi di testo sarà sufficiente dichiarare il connettore (tramite il builder) nel metodo *onCreate()* di *ConnectorActivity*, per come è implementato, il sistema provvederà ad aggiungerlo nelle opportune *RecyclerView* e a costruire il comportamento di inserimento dei parametri e loro archiviazione nel database. Per connettori con input differenti da normali field testuali sarà necessario, oltre ad effettuare il suddetto processo, creare una activity che lo gestisca.

7.3 Skill

7.3.1 Implementazione nuovi connettori

Per aggiungere un nuovo connettore è necessario farlo ereditare da */src/actions/Action.js* e assegnargli un nome identificativo non già usato. Il costruttore necessita solo di due parametri che passerà al padre. Il fulcro del connettore sta nel metodo asincrono *run()*, esso deve ritornare un oggetto **check** composto da due valori: **output** che contiene l'effettivo testo che Amazon Alexa leggerà e **slotReq**. Questo secondo valore, che normalmente è settato a 'DEFAULT', serve nel caso l'action necessiti di un'interazione con l'utente, per esempio chiedergli un parametro, e contiene il nome dello slot da richiedere.

7.3.2 Connettori che interagiscono con l'utente

Nel caso in cui **slotReq** non sia settato a 'DEFAULT', la Skill richiamerà l'action specificata finchè *slotReq* non tornerà al valore normale, inserendo i nuovi valori in coda all'array **param[]**. Inoltre, nel caso in cui lo sviluppatore necessiti di nuovi slot, esso dovrà aggiungerli alla build della Skill tramite console Amazon Developer Alexa.

Si sconsiglia l'utilizzo di uno slot generico e polivalente, perchè azioni diverse probabilmente necessitano di tipologie di input diverse. Dalla console di Alexa, è possibile stabilire la tipologia di input di uno slot, scegliendo dalle librerie predefinite di Amazon oppure creando uno slot custom. Questo permette ad Alexa di migliorare la comprensione dell'iterazione con l'utente, in quanto sarà già a conoscenza del formato o delle possibili risposte che l'utente potrà fornire. Consigliamo nel caso di slot contenenti testi molto lunghi di chiedere conferma all'utente tramite slot **confirmationSlot** e controllare se la risposta dell'utente sia stata "si" o "no".

7.3.3 Collegare il connettore alla Skill

Infine dopo aver aggiunto eventuali slot dalla Console si deve collegare il connettore andando ad aggiungere il nome univoco dello stesso nel file `/src/Utils/ActionFactory.js` all'interno dello switch, facendo tornare un oggetto del tipo del connettore da aggiungere.

8 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.