

# MegAlexa

Aricchitore di skill di Amazon Alexa

---

## MANUALE DELLO SVILUPPATORE

GRUPPO ZEROSEVEN



<b>Versione</b>	1.0.0
<b>Data Redazione</b>	2019-03-29
<b>Redazione</b>	Mirko Franco Stefano Zanatta
<b>Verifica</b>	Gian Marco Bratzu
<b>Approvazione</b>	Andrea Deidda
<b>Uso</b>	Esterno
<b>Distribuzione</b>	Prof. Tullio Vardanega Prof. Riccardo Cardin Gruppo ZeroSeven Zero12 s.r.l.
<b>Email di contatto</b>	zerosevnswe@gmail.com

## Registro delle modifiche

Versione	Data	Descrizione	Autore	Ruolo
1.0.0	2019-05-10	Approvazione documento per rilascio RA	Andrea Deidda	Responsabile
0.3.0	2019-05-10	Verifica documento	Gian Marco Bratzu	Verificatore
0.2.2	2019-05-09	Stesura §A	Mirko Franco	Analista
0.2.1	2019-05-07	Stesura capitolo §5	Stefano Zanatta	Analista
0.2.0	2019-04-10	Approvazione documento per rilascio RQ	Ludovico Brocca	Responsabile
0.2.0	2019-04-10	Verifica documento	Mirko Franco	Verificatore
0.1.1	2019-04-09	Stesura §5, §4.1	Stefano Zanatta	Progettista
0.1.0	2019-04-09	Verifica §2.1, §3, §2.3	Gian Marco Bratzu	Progettista
0.0.6	2019-04-08	Stesura §3.2.1, §2.3	Stefano Zanatta	Progettista
0.0.5	2019-04-05	Stesura capitolo §3	Mirko Franco	Progettista
0.0.4	2019-04-05	Stesura capitolo §2.1	Mirko Franco	Progettista
0.0.3	2019-03-30	Stesura capitolo §1	Mirko Franco	Progettista
0.0.2	2019-03-30	Stesura struttura documento	Mirko Franco	Progettista
0.0.1	2019-03-29	Creo documento	Mirko Franco	Progettista

# Indice

<b>1</b>	<b>Introduzione</b>	<b>4</b>
1.1	Scopo del documento . . . . .	4
1.2	Scopo del prodotto . . . . .	4
1.3	Glossario . . . . .	4
<b>2</b>	<b>Procedura di installazione</b>	<b>5</b>
2.1	Requisiti di Sistema . . . . .	5
2.1.1	Applicazione . . . . .	5
2.1.2	Skill . . . . .	5
2.2	Installazione App . . . . .	5
2.3	Installazione Skill . . . . .	7
<b>3</b>	<b>Tecnologie utilizzate</b>	<b>8</b>
3.1	Amazon Web Service . . . . .	8
3.1.1	AWS DynamoDB . . . . .	8
3.1.2	AWS Lambda . . . . .	8
3.1.3	AWS API Gateway . . . . .	9
3.1.4	AWS CloudWatch . . . . .	9
3.1.5	Kotlin . . . . .	9
3.1.6	XML . . . . .	10
3.1.7	Node.js . . . . .	10
3.1.8	Npm . . . . .	10
3.1.9	7z . . . . .	10
3.2	Dipendenze esterne . . . . .	10
3.2.1	Alexa Skill . . . . .	10
3.2.2	App Android . . . . .	11
<b>4</b>	<b>Architettura</b>	<b>12</b>
4.1	Skill . . . . .	12
4.1.1	Index . . . . .	14
4.1.2	Classe User . . . . .	14

4.1.3	Classe Workflow . . . . .	15
4.1.4	Package services . . . . .	15
4.1.5	Package blocks . . . . .	16
4.1.5.1	Package blocks.utils . . . . .	16
4.1.6	Package connectors . . . . .	16
4.2	App . . . . .	18
<b>5</b>	<b>Estensione delle funzionalità</b>	<b>28</b>
5.1	Estensione App . . . . .	28
5.2	Estensione Skill (lambda) . . . . .	29
5.2.1	Nuovo blocco . . . . .	29
5.2.2	Nuove frasi per Alexa . . . . .	30
5.2.3	Nuovo metodo d'accesso al database . . . . .	30
5.2.4	Nuovo sistema di autenticazione . . . . .	31
<b>A</b>	<b>Glossario</b>	<b>32</b>

# Elenco delle figure

2.1	<i>Tasti Build e Run</i> . . . . .	6
4.1	<i>Alexa developer platform</i> . . . . .	12
4.2	<i>Skill class diagram - Package</i> . . . . .	13
4.3	<i>Skill class diagram - User</i> . . . . .	15
4.4	<i>Skill class diagram - Workflow</i> . . . . .	15
4.5	<i>Skill class diagram - WorkflowService</i> . . . . .	15
4.6	<i>Skill class diagram - Blocks package</i> . . . . .	16
4.7	<i>Skill class diagram - blocks package</i> . . . . .	17
4.8	<i>App class diagram - blocks package</i> . . . . .	19
4.9	<i>App class diagram - MVVM</i> . . . . .	20
4.10	<i>App class diagram - Blocks</i> . . . . .	21
4.11	<i>App class diagram - Service</i> . . . . .	22
4.12	<i>App class diagram - UI</i> . . . . .	24
4.13	<i>App class diagram - ViewModel</i> . . . . .	25
4.14	<i>App class diagram - BlockConnection</i> . . . . .	26
4.15	<i>App class diagram - Model</i> . . . . .	27

# Capitolo 1

## Introduzione

### 1.1 Scopo del documento

Il presente documento vuole essere una guida introduttiva del software *MegAlexa*, indirizzata agli sviluppatori che volessero adattarlo o estenderlo. Vengono spiegate le tecnologie interessate, l'architettura in dettaglio e le possibilità di estensione.

### 1.2 Scopo del prodotto

Lo scopo del progetto è lo sviluppo di un applicativo mobile in grado di creare delle routine personalizzate per gli utenti gestibili tramite Alexa di *Amazon<sub>G</sub>*. L'obiettivo è la creazione di una *skill<sub>G</sub>* in grado di avviare *workflow<sub>G</sub>* creati dagli utenti fornendogli dei *connettori<sub>G</sub>*.

### 1.3 Glossario

Volendo evitare incomprensioni ed equivoci per rendere la lettura del documento più semplice e chiara si riporta in appendice un glossario nel quale sono contenute le definizioni dei termini tecnici, dei vocaboli ambigui, degli acronimi e delle abbreviazioni. Questi termini sono evidenziati nel documento con una G al pedice (esempio: *Glossario<sub>G</sub>*).

# Capitolo 2

## Procedura di installazione

### 2.1 Requisiti di sistema

*MegAlexa* è composta da un'applicazione compatibile con la maggior parte dei dispositivi *Android<sub>G</sub>* e da una skill Alexa.

#### 2.1.1 Applicazione

L'applicazione è compatibile con tutti i dispositivi Android con versione 4.4 o superiore. Per poter modificare e ampliare la app i seguenti requisiti devono essere soddisfatti:

- **IDE Android Studio<sub>G</sub>**: necessaria per eseguire e testare la app nel corso del suo sviluppo;
- **Git<sub>G</sub>**: necessario per effettuare il `clone` della repository e il versionamento successivo del codice;
- **Gradle<sub>G</sub>**: per il download automatico delle dipendenze e la compilazione del codice (consigliata versione 4.10.0 o superiore).

#### 2.1.2 Skill

La skill è compatibile con tutti i dispositivi Amazon *Echo<sub>G</sub>*.

### 2.2 Installazione app

Per installare la app è obbligatorio seguire i seguenti passi:

1. **Acquisire la repository:** eseguire il comando `git clone` seguito dal seguente URL: <https://github.com/sgt390/ProgettoSweCodice.git>;
2. **Registrare il dispositivo:** accedere alla console di sviluppo offerta da amazon con le credenziali inviate dai membri del gruppo e registrare una key univoca per la applicazione denominata *MegAlexa*<sup>1</sup>;
3. **Posizionamento della key:** una volta aperto Android Studio, creare una cartella nel percorso `/MegAlexa/app/src/main`, nominarla `assets`, creare un file di testo nominato `api_key.txt` e posizionarlo nella cartella appena creata;
4. **Gradle Sync:** a questo punto, se la procedura è stata eseguita correttamente, Gradle dovrebbe scaricare in automatico le dipendenze per l'avvio della app, nel caso in cui ciò non avvenga, eseguire il comando `./gradlew build` nella cartella di root del progetto;
5. **Compilazione ed esecuzione:** la compilazione può avvenire mediante il comando `./gradlew build` oppure mediante la pressione dell'icona a forma di martello presente in alto su Android Studio (vedi figura 2.1), l'esecuzione avviene alla pressione del tasto run presente nell'IDE(vedi figura 2.1).  
Alla prima esecuzione verrà richiesta l'installazione di una versione Android per l'emulatore: scegliere l'opzione più gradita e continuare.  
In alternativa, è possibile eseguire l'applicazione su un dispositivo Android predisposto correttamente(per maggiori dettagli visitare <https://developer.android.com/training/basics/firstapp/running-app>).



Figura 2.1: *Tasti Build e Run*

<sup>1</sup><https://developer.amazon.com/loginwithamazon/console/site/lwa/overview.html>



## 2.3 Installazione skill

Per ognuno dei seguenti comandi è richiesta l'installazione del package manager **npm**. Installazione della skill e delle sue dipendenze:

- clonare la repository attraverso il comando *git clone*  
*<https://github.com/sgt390/MegAlexaSkill/>*;
- eseguire il comando *npm install* per installare automaticamente le dipendenze.

Pubblicare la skill in AWS Lambda:

- installare il programma *7zG* e inserire il suo eseguibile tra le variabili di sistema;
- installare e configurare *aws-cli*<sup>2</sup>;
- da terminal, eseguire il comando *npm run publish-lambda*.

Eseguire i test di unità:

- da terminal, eseguire il comando *npm run unitTest*.

Eseguire i test di integrazione:

- da terminal, eseguire il comando *npm integrationTest*.

---

<sup>2</sup><https://aws.amazon.com/it/cli/>

# Capitolo 3

## Tecnologie utilizzate

### 3.1 Amazon Web Service

Amazon Web Service è una piattaforma di cloud computing sicura che offre servizi di calcolo, memorizzazione, distribuzione di contenuti e altre funzionalità per aiutare il business ad essere scalabile e crescere con facilità.

AWS fornisce infatti prodotti e servizi per costruire applicazioni, anche sofisticate, in modo flessibile, scalabile, economico e con un'ottima resistenza ai guasti.

#### 3.1.1 AWS DynamoDB

Amazon *DynamoDB<sub>G</sub>* è un database che supporta i modelli di dati di tipo documento e di tipo chiave-valore che offre prestazioni di pochi millisecondi a qualsiasi livello. Si tratta di un database multi master, multi regione e completamente gestito che offre sicurezza integrata, backup, ripristino e cache in memoria per applicazioni Internet. DynamoDB può gestire oltre 10 trilioni di richieste al giorno e supporta picchi di oltre 20 milioni di richieste al secondo.

#### 3.1.2 AWS Lambda

*AWS Lambda<sub>G</sub>* consente di eseguire codice senza dover effettuare il provisioning né gestire il server. Le tariffe sono calcolate in base ai tempi di elaborazione.

Con Lambda, è possibile eseguire codice per qualunque tipo di applicazione o di servizio back-end, senza alcuna amministrazione. Una volta caricato il codice Lambda si prende carico delle azioni necessarie per eseguirlo e ricalibrarne le risorse con la massima disponibilità.

È possibile configurare il codice in modo che venga attivato automaticamente da altri servizi AWS oppure che venga richiamato direttamente da qualsiasi app Web o mobile.

### 3.1.3 AWS API Gateway

*AWS API Gateway<sub>G</sub>* è un servizio completamente gestito che semplifica la creazione, la pubblicazione, la manutenzione e la protezione delle API su larga scala. Con semplicità è possibile creare e configurare API REST che fungano da "porta di ingresso" per le applicazioni, per consentire l'accesso ai dati, alla logica di business o alle funzionalità dai propri servizi back-end. API Gateway gestisce tutte le attività di accettazione ed elaborazione relative a centinaia di migliaia di chiamate ad *API<sub>G</sub>* simultanee, inclusi gestione del traffico, controllo di accessi e autorizzazioni, monitoraggio e gestione delle versioni delle API Gateway non prevede alcuna tariffa minima né investimenti iniziali. Vengono addebitati solo i costi di chiamate API ricevute e i volumi di dati trasferiti in uscita e con il modello tariffario a scaglioni di API Gateway potrai ridurre i costi al variare dell'utilizzo delle *API<sub>G</sub>*.

### 3.1.4 AWS CloudWatch

AWS CloudWatch è un servizio di monitoraggio e gestione creato per gli sviluppatori, operatori di sistema, ingegneri responsabili del sito e manager IT. CloudWatch fornisce dati e analisi concrete per monitorare le applicazioni, capire e rispondere ai cambiamenti di prestazioni a livello di sistema, ottimizzare l'utilizzo delle risorse e ottenere una visualizzazione unificata dello stato di integrità operativa.

### 3.1.5 Kotlin

*Kotlin<sub>G</sub>* è un linguaggio di programmazione general purpose, multi-paradigma, open source sviluppato dall'azienda di software JetBrains.

Kotlin si basa sulla JVM (Java Virtual Machine) ed è ispirato ad altri linguaggi di programmazione tra i quali Scala e lo stesso Java, mentre ulteriori spunti sintattici sono stati presi da linguaggi classici, come il Pascal e moderni come Go o F#.

Il vantaggio maggiore rispetto a Java risulta essere la sintassi più leggibile, insieme alla possibilità di eseguire file Java all'interno di progetti Kotlin (visto che entrambi utilizzano la JVM per produrre il bytecode).

### 3.1.6 XML

XML è un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio marcatore basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.

Android fornisce gli schema necessari per comporre widget e posizionarli sullo schermo, separando ulteriormente la parte logica da quella grafica.

### 3.1.7 Node.js

La skill è sviluppata attraverso un progetto Node.js. Tutte le classi (apparte `index.js`) sono scritte in *TypeScript<sub>G</sub>*, ma vengono compilate in Javascript prima di essere trasferite in AWS Lambda.

### 3.1.8 Npm

Gestore di package per node.js. Permette l'esecuzione di comandi personalizzati (come descritto in §2.3).

### 3.1.9 7z

Permette di comprimere i file da linea di comando. È richiesto dalla procedura di build della skill.

## 3.2 Dipendenze esterne

### 3.2.1 Alexa skill

Tutte le dipendenze principali si trovano nel file *package.json*, mentre quelle secondarie (cioè le dipendenze delle dipendenze) stanno nel file *package-lock.json*.

Dipendenze principali:

- **ask-sdk:** (Alexa Skill Kit) permette di comunicare direttamente con i servizi Amazon Alexa;
- **axios:** chiamate HTTP. Utilizzato principalmente in *WorkflowService* per le chiamate REST e nei *connettori<sub>G</sub>*;
- **openweather-apis:** informazioni riguardanti il tempo atmosferico;

- **rss-parser:** trasformazione di un file RSS in un testo più leggibile;
- **twitter:** comunicazione con i servizi Twitter.

Dipendenze utili solo allo sviluppatore (test e compilazione):

- **mocha:** framework per i test in Javascript;
- **chai:** funzionalità aggiuntive mocha;
- **@types/node:** definizioni dei tipi di Node.js per *TypeScript*<sub>G</sub>;
- **@types/mocha:** adattatore mocha per TypeScript;
- **@types/chai:** adattatore chai per TypeScript;
- **@types/chai-as-promised:** dipendenza aggiuntiva di chai per valutare le *promise*;
- **typescript:** permette la compilazione da TypeScript a JavaScript.

### 3.2.2 App android

Tutte le dipendenze si trovano nel file *build.gradle(Module:app)*.

Dipendenze principali:

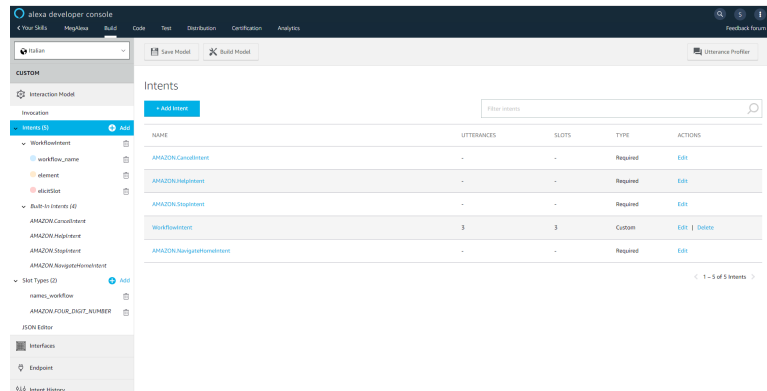
- **kotlin-stdlib-jdk7** Libreria standard di *Kotlin*<sub>G</sub> che si appoggia automaticamente alla versione più aggiornata;
- **Anko** Libreria utilizzata per le chiamate asincrone;
- **openweather-apis:** informazioni riguardanti il tempo atmosferico;
- **rss-parser:** trasformazione di un file RSS in un testo più leggibile;
- **TwitterCore:** comunicazione con i servizi Twitter;
- **Lifecycle:** permettono un'implementazione semplice al pattern observer;
- **AppCompatActivity:** Libreria più stabile per le interfacce grafiche su *Kotlin*<sub>G</sub>;
- **JUnit** framework per i test in Kotlin;
- **login-with-amazon-sdk.jar** Utilizzato per i servizi di login forniti da Amazon.

# Capitolo 4

## Architettura

### 4.1 Skill

La  $skill_G$  è reperibile al seguente link: <https://developer.amazon.com/alexa/> (è necessario un account developer Amazon per accedervi). La skill è rappresentata da un JSON, configurabile attraverso l'UI fornita dalla piattaforma Alexa developer.



**Figura 4.1:** *Alexa developer platform*

La logica della skill si trova in una  $AWS\ Lambda_G$ , quindi è serverless. Questo significa che viene creata una nuova istanza della Skill ogni volta che arriva una richiesta da Alexa (ogni utente genera una istanza diversa). L'architettura della Skill si basa sul concetto di avere una skill leggera, che esegue poche operazioni ogni volta che questa viene invocata. Una richiesta da parte di Alexa viene catturata dall'index, che la elabora e ritorna un risultato. Una richiesta è definita da un JSON contenente molteplici informazioni sullo stato del dialogo, l'utente, errori e molto altro. Una

dettagliata descrizione si trova sulla documentazione di Amazon AWS<sup>1</sup>. La lingua della skill viene impostata automaticamente a seconda della richiesta. Le frasi che Alexa può dire si trovano nei file di configurazione `phrases-EN.json` e `phrases-IT.json`. Le informazioni più importanti contenute nel file `JSON_G` sono:

- **`handlerInput.requestEnvelope.request.type`**: rappresenta il tipo di richiesta: `IntentRequest` o `LaunchRequest`. `LaunchRequest` rappresenta la prima iterazione con la skill ("Alexa, apri MegAlexa"), `IntentRequest` rappresenta tutte le altre richieste;
- **`handlerInput.requestEnvelope.request.intent.name`**: contiene il nome della richiesta, definiti dove la skill si trova. I casi più comuni sono `"WorkflowIntent"`, `"StopIntent"` e `"HelpIntent"`;
- **`handlerInput.requestEnvelope.request.slots`**: contiene una lista di slot, cioè dei parametri che l'utente deve dire per continuare con il dialogo con Alexa;
- **`handlerInput.requestEnvelope.request.attributesManager`**: permette la gestione degli attributi di sessione. Servono per salvare delle variabili da una chiamata all'altra della skill (lambda).

Il seguente diagramma dei package descrive le dipendenze ad alto livello della Skill.

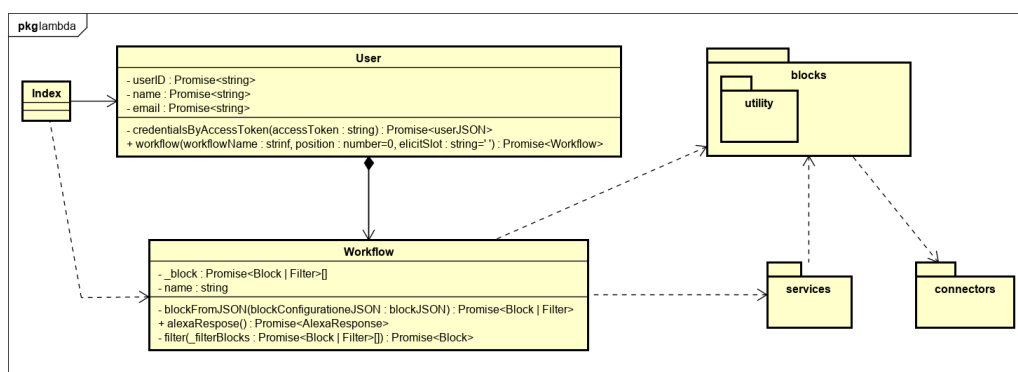


Figura 4.2: Skill class diagram - Package

<sup>1</sup><https://developer.amazon.com/docs/custom-skills/request-and-response-json-reference.html>

La classe *User* rappresenta un singolo utente che sta utilizzando la skill. L'utente usa la classe *WorkflowService* per creare un *Workflow*. *Workflow* crea una lista di *Block*. Alcuni *Block* creano un *connector* per fare chiamate HTTP.

I blocchi che devono fare chiamate HTTP, vengono rappresentati come delle Promise, per questo motivo anche *Workflow* viene rappresentato in una promise in *User*.

Tutte le classi sono scritte in *TypeScript<sub>G</sub>*. L'index è scritto in Javascript (per compatibilità con ask-sdk). Prima di fare il deploy in *AWS Lambda<sub>G</sub>*, i file *TypeScript* vengono compilati in *JavaScript*.

### 4.1.1 Index

Index contiene gli handler, funzioni che catturano le richieste da parte di Alexa. Tutti gli handler sono definiti nella funzione "handler" e vengono eseguiti nell'ordine di dichiarazione.

Ogni singolo handler è formato da due parti:

- **canHandle:** verifica se quello è il giusto handler da eseguire. Il controllo viene fatto usando i parametri definiti in §4.1. Se nessun handler può gestire la richiesta, viene invocato *Error handler*;
- **handle:** se *canHandle* ritorna True, questa è la funzione che viene eseguita e deve ritornare una risposta comprensibile da Alexa.

### 4.1.2 Classe User

In *User* sono presenti i seguenti metodi:

- **credentialsByAccessToken:** prende come parametro un access token (fornito da Amazon Alexa) e ritorna una Promise contenente un JSON, il quale contiene username, email e userID. Lo userID è lo stesso di quello ottenuto attraverso il collegamento dell'applicazione all'applicazione *Android<sub>G</sub>*, permettendo di autenticare l'utente nel database;
- **workflow:** usa *WorkflowService* per creare un *Workflow* a partire dalla sua rappresentazione in formato *JSON<sub>G</sub>*.



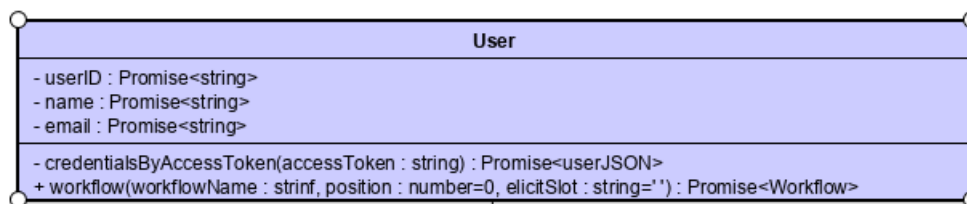


Figura 4.3: Skill class diagram - User

### 4.1.3 Classe Workflow

In *Workflow* sono presenti i seguenti metodi:

- **blockFromJSON**: crea un blocco a partire dalla sua rappresentazione in JSON;
- **alexaResponse**: crea la risposta sotto forma di `Promise<string>` a partire dalle risposte di ogni blocco;
- **filter**: rimuove tutti i *Filter* dalla lista di blocchi, e filtra i *Block* (chiamando l'apposito metodo nel *Block*) che seguono direttamente ogni Filter. Questo metodo non genera side-effect.

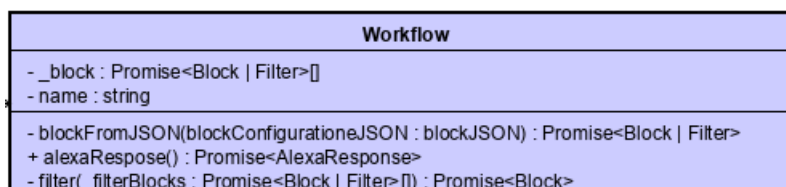


Figura 4.4: Skill class diagram - Workflow

### 4.1.4 Package services

Il package services contiene la classe *WorkflowService*, che si occupa di creare un *Workflow*, dopo aver ottenuto la sua rappresentazione  $JSON_G$  dal database (con una chiamata REST).

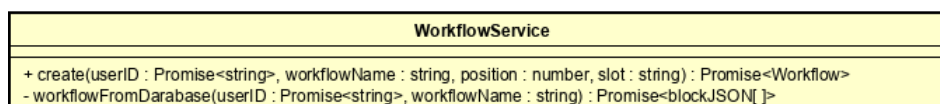


Figura 4.5: Skill class diagram - WorkflowService

### 4.1.5 Package blocks

Il package blocks contiene tutti i blocchi della *skill<sub>G</sub>* e il package utils. I blocchi implementano tutti l'interfaccia *Block*, alcuni implementano anche *Filterable* e *ElicitBlock*.

Un blocco Filterable può essere rappresentato come una lista, quindi deve permettere di ritornare una versione di se stesso con meno elementi.

Un blocco ElicitBlock richiede all'utente dei parametri aggiuntivi per poter essere eseguito.

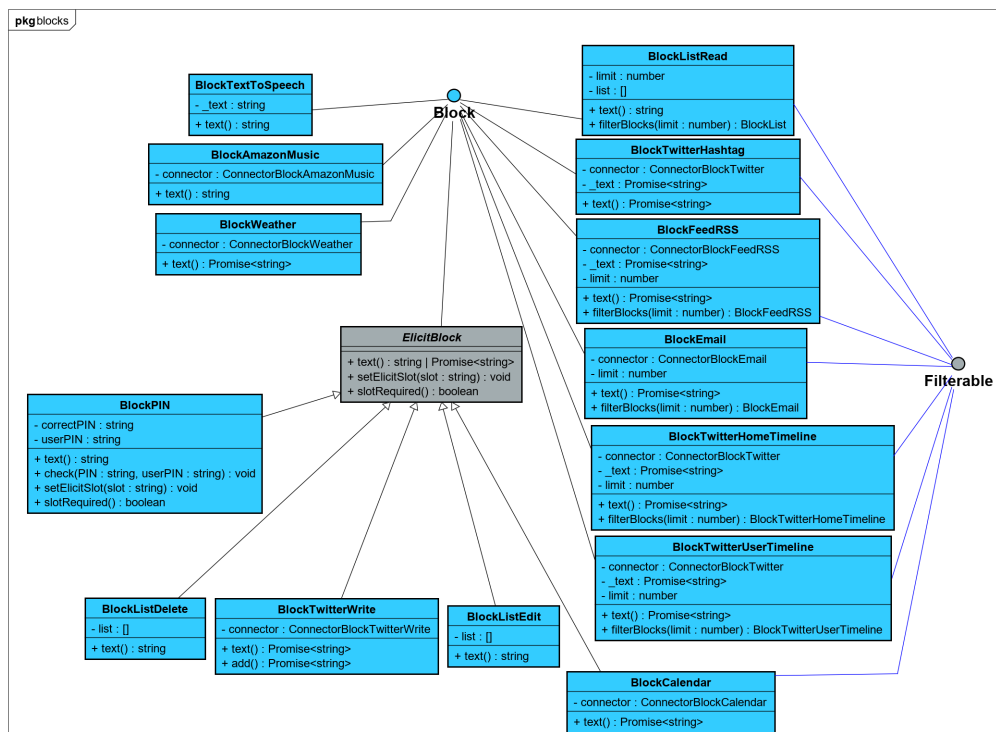


Figura 4.6: Skill class diagram - Blocks package

#### 4.1.5.1 Package blocks.utils

Il package *blocks.utils* contiene le interfacce utili ai *Block* (*ElicitBlock* e *Filterable*).

### 4.1.6 Package connectors

Il package *connectors* contiene i connettori utilizzati dai blocchi.

Un Connector permette al blocco di ottenere le informazioni che gli servono

da internet. Per esempio, BlockWeather (un blocco che rappresenta il meteo) chiamerà una libreria per conoscere il meteo di una certa zona. Ogni *Connector* deve processare il risultato e trasformarlo nel testo che Alexa dovrà ripetere.

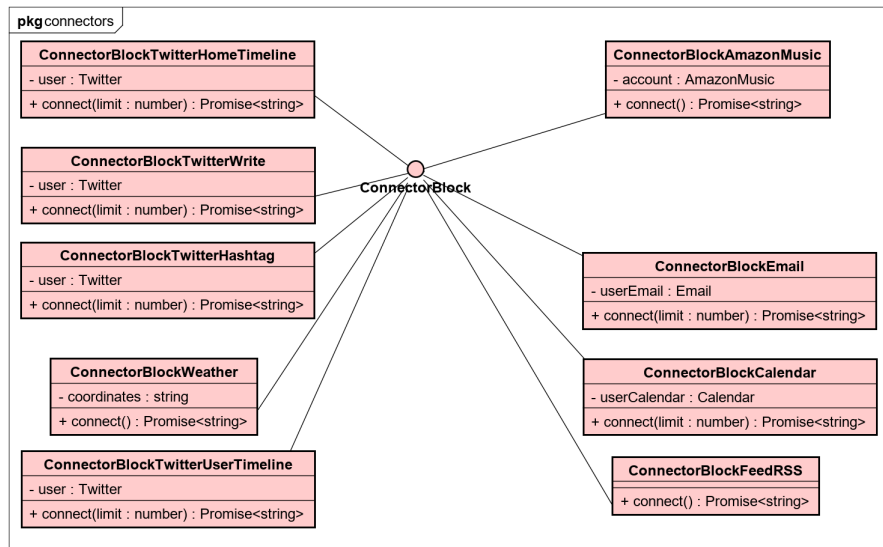


Figura 4.7: Skill class diagram - blocks package

## 4.2 App

L'app si trova nella repository GitHub: <https://github.com/sgt390/ProgettoSweCodice>.  
I pattern che abbiamo utilizzato sono:

- **ModelViewViewModel** utilizzato per la gestione e scambio dati tra Model e View;
- **Builder** utilizzato per il Model (Workflow);
- **Singleton** utilizzato per il Model (Workflow);
- **Facade** utilizzato per le connessioni ad  $API_G$  esterne;
- **Strategy** utilizzato per il service;
- **Template Method** utilizzato per il service.

Nella seguente immagine viene descritta l'architettura dell'app. La struttura è definita da tre cartelle:

- **model:** contiene il modello delle classi sviluppate;
- **ui:** definisce la user interface in cui sono contenute le classi grafiche dell'applicazione;
- **util:** in cui sono contenute le funzionalità utilizzate per le chiamate esterne all'applicazione e il design pattern per lo scambio di informazioni tra modello e interfaccia grafica.

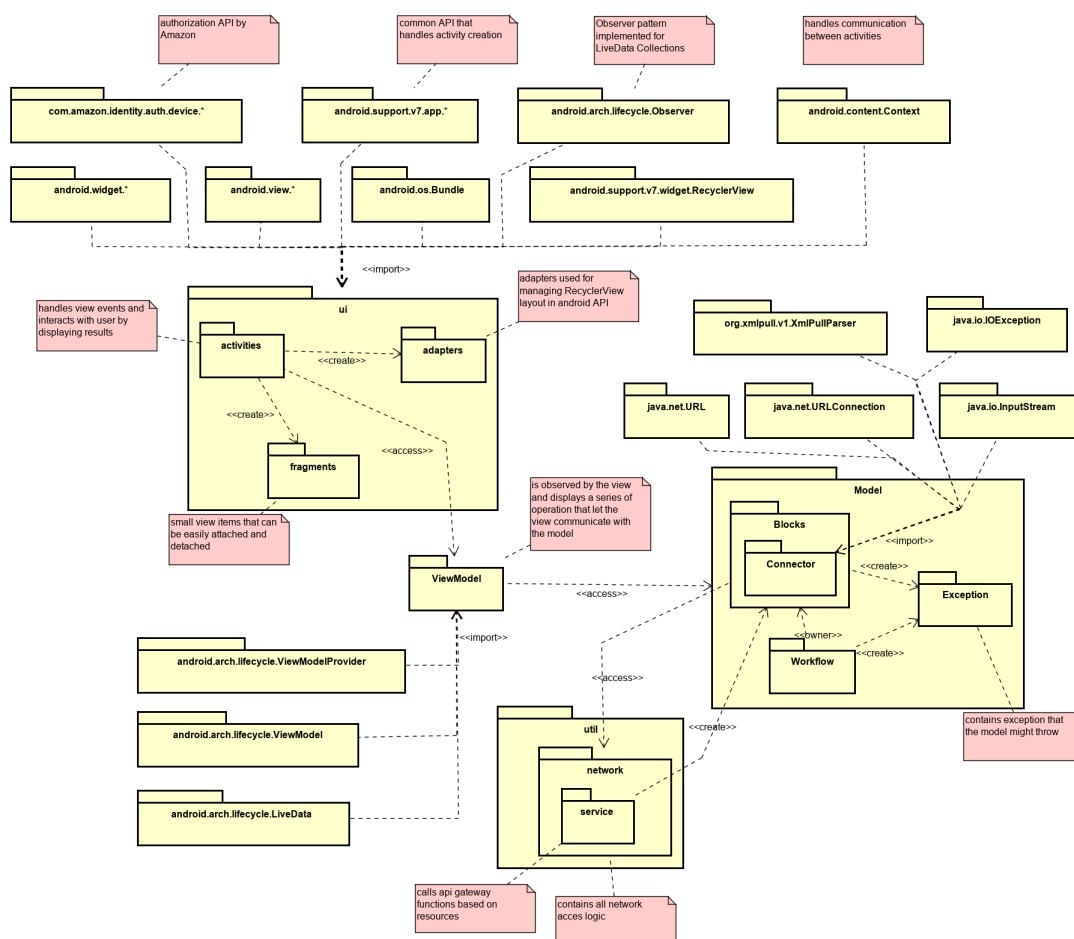


Figura 4.8: App class diagram - blocks package

La seguente immagine descrive la progettazione del pattern  $MVVM_G$ . Il ViewModel ha lo scopo di utilizzare l'istanza di Megalexa per fornire i contenuti personali dell'utente all'interfaccia grafica.

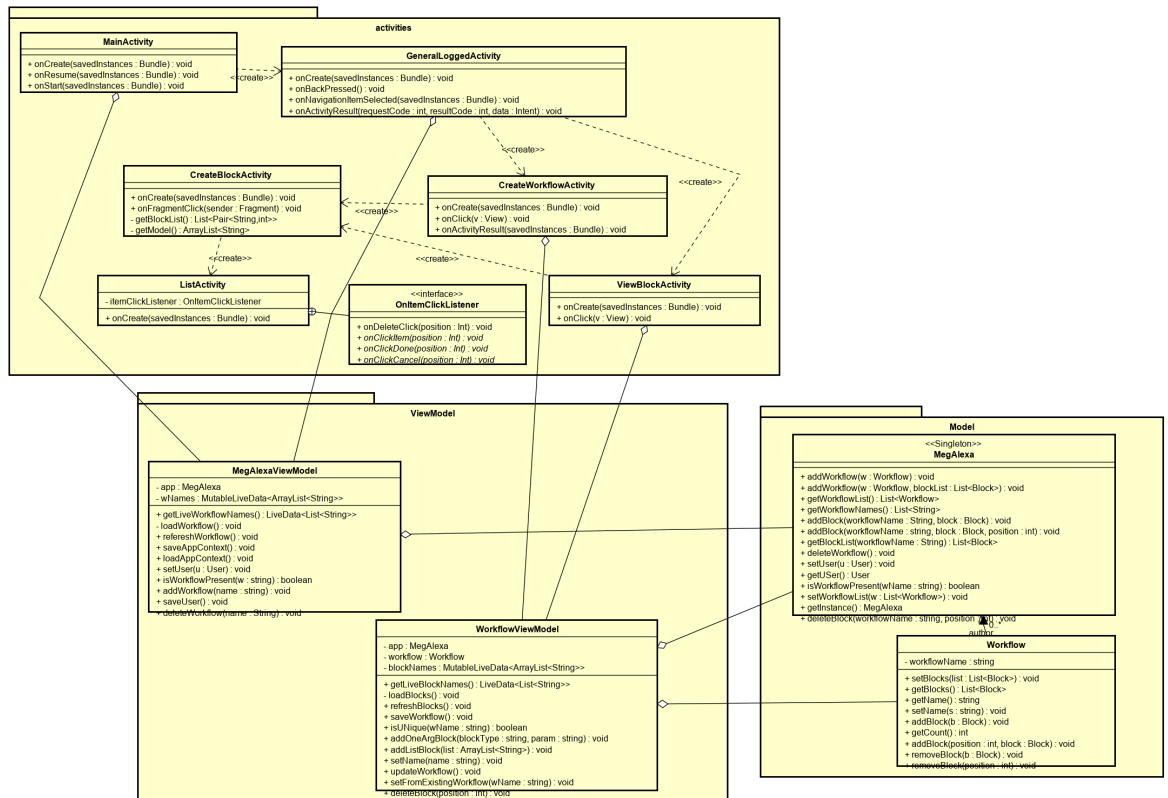


Figura 4.9: App class diagram - MVVM

La seguente immagine descrive la progettazione dei blocchi. Tutti i blocchi implementano l'interfaccia `Block` che ha come unico metodo `getInformation()` per ricavare informazioni aggiuntive sul blocco. I blocchi per i quali è necessario filtrare la quantità di risultati devono estendere l'interfaccia *Filtrable* contenuta nella cartella stessa.

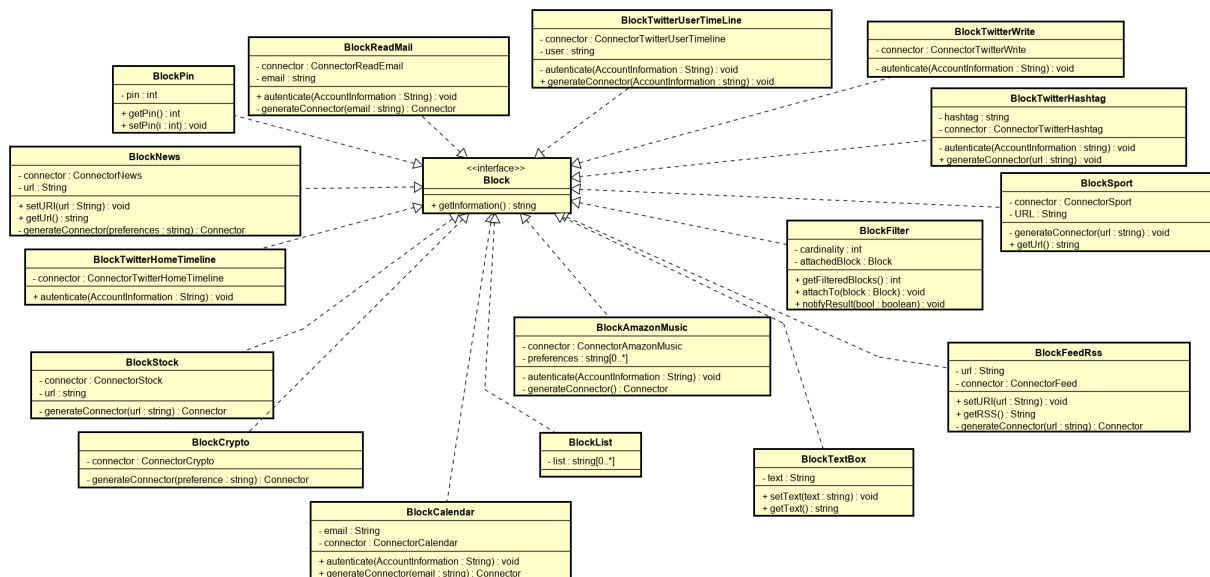


Figura 4.10: App class diagram - Blocks

La seguente immagine descrive la progettazione dei service. I service sono utilizzati per le chiamate REST e per la conversione dei blocchi tramite i metodi:

- **convertToJSON(t: T):** utilizzato per la conversione delle informazioni ottenute in  $JSON_G$  e successivamente salvate in database;
- **convertFromJSON(J: JSONObject):** utilizzato per la conversione delle informazioni ottenute dal database in blocchi dati e successivamente mostrati all'utente.

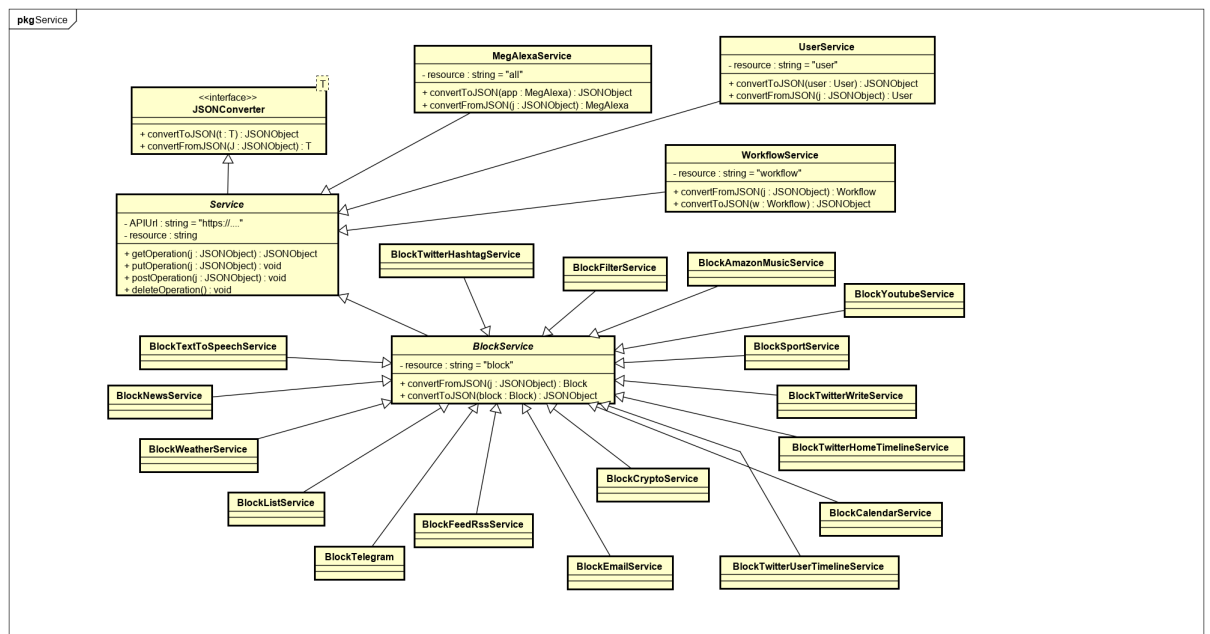


Figura 4.11: App class diagram - Service



La seguente immagine rappresenta la progettazione dell'UI.  
L'applicazione utilizza le seguenti *activities<sub>G</sub>*:

- **MainActivity:** permette all'utente di autenticarsi;
- **GeneralLoggedActivity:** permette all'utente di visualizzare i propri *workflow<sub>G</sub>*;
- **CreateWorkflowActivity:** permette all'utente di creare un nuovo workflow;
- **ViewBlockActivity:** permette all'utente di visualizzare e modificare i blocchi presenti all'interno di un workflow;
- **CreateBlockActivity:** permette all'utente di aggiungere nuovi blocchi.

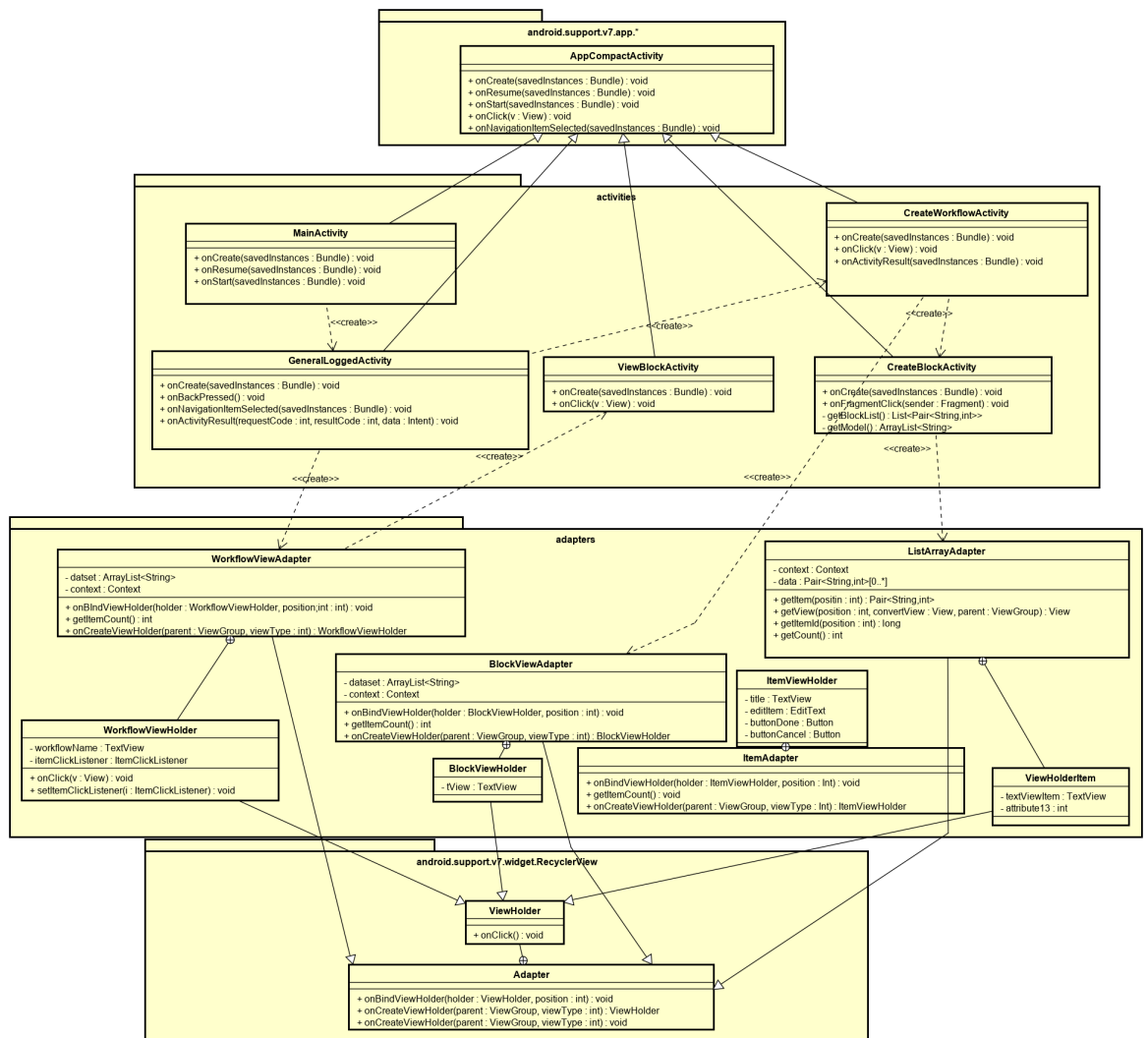


Figura 4.12: App class diagram - UI

La seguente immagine descrive la progettazione del ViewModel.

- **MegAlexaViewModel:** si occupa di gestire le informazioni tra interfaccia e modello relativo ai *workflow<sub>G</sub>*;
- **WorkflowViewModel:** si occupa di gestire le informazioni tra interfaccia e modello relativo ai blocchi.

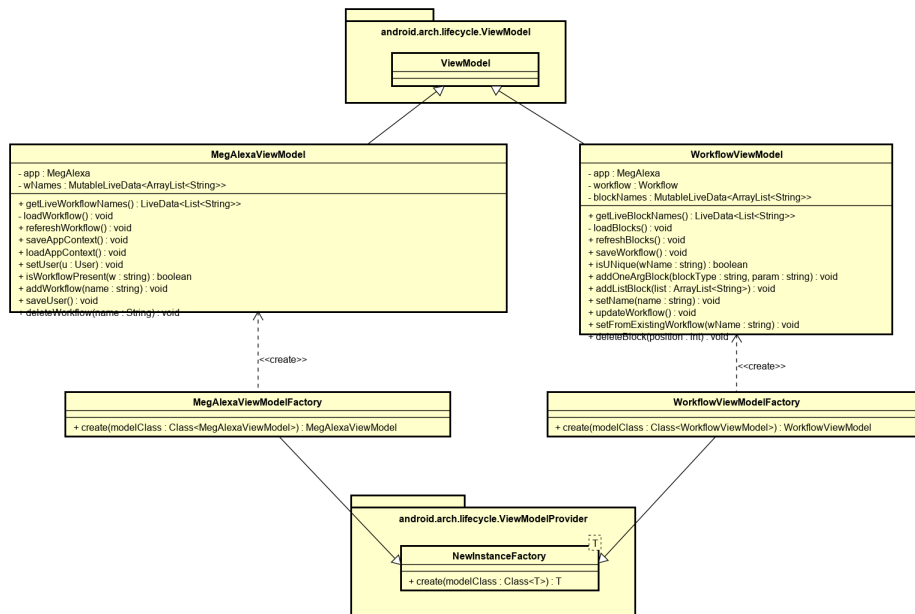


Figura 4.13: App class diagram - ViewModel

La seguente immagine descrive la progettazione dei blocchi che implementano i *connettori*<sub>G</sub>. I connettori sono utilizzati per la gestione e validazione dei dati ottenuti da chiamate esterne all'applicazione.

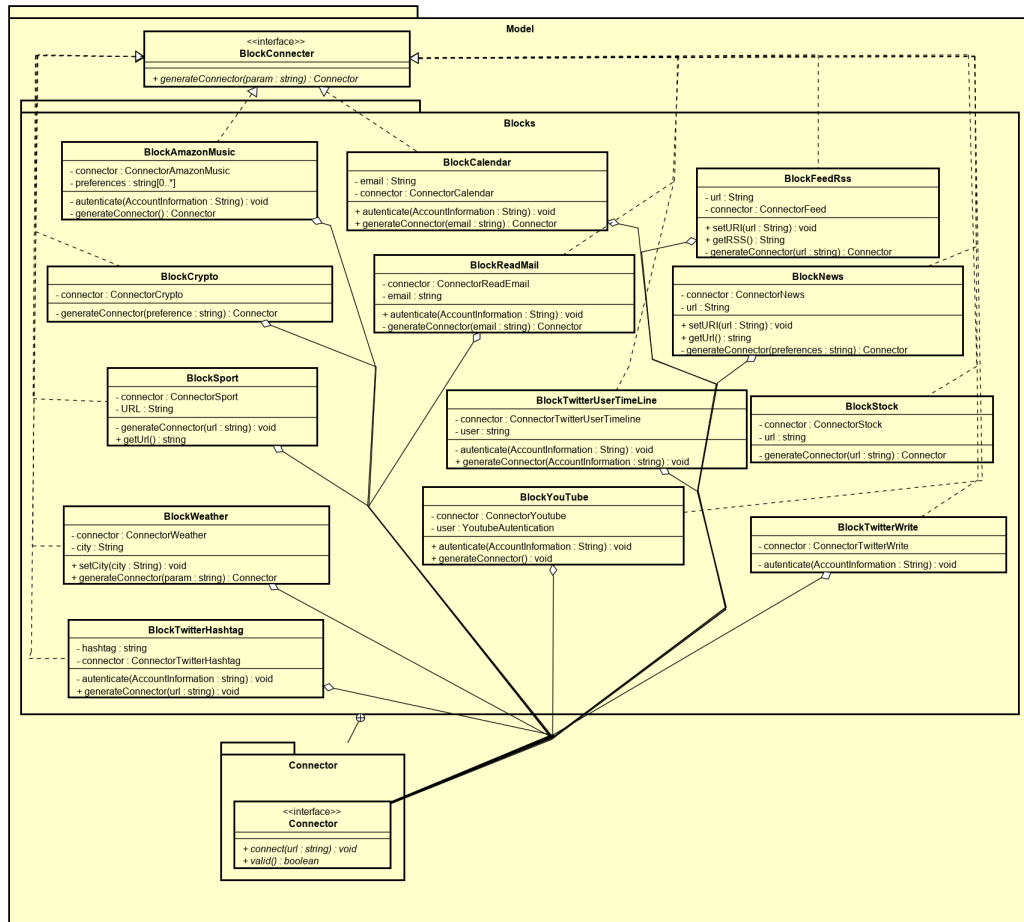


Figura 4.14: App class diagram - BlockConnection

La seguente immagine descrive la progettazione del Model che contiene tutti i dati inseriti dall'utente.

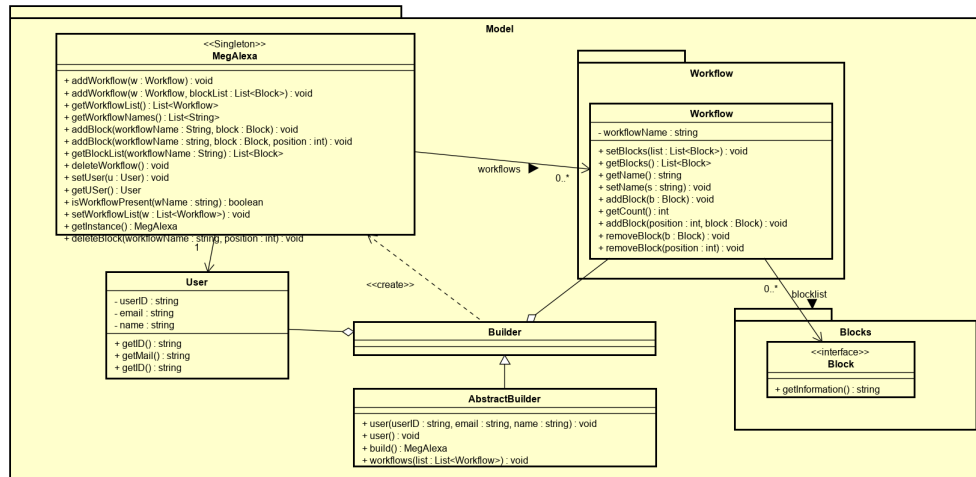


Figura 4.15: App class diagram - Model

# Capitolo 5

## Estensione delle funzionalità

In questo capitolo vengono descritti i passaggi per l'estensione del prodotto *MegAlexa*.

### 5.1 Estensione app

L'estensione della app avviene mediante l'aggiunta di nuovi blocchi. Segue una lista di attività da attuare che descrive i passi da eseguire per un'estensione corretta:

- Definire un blocco che derivi dall'interfaccia **Block** e, se necessario, l'interfaccia **Filtrable** posta nel package **blocks**;
- Se il blocco richiede una validazione tramite chiamata esterna, definire un *connettore<sub>G</sub>* che estenda **Connector** nel package **connectors**;
- Se la chiamata esterna prevede l'utilizzo di una *API<sub>G</sub>* particolare, definire tali interazioni incapsulandole in una nuova classe da posizionare nel package **network**;
- Definire le conversioni da *JSON<sub>G</sub>* e a JSON dichiarando una nuova classe che derivi da **BlockService** nel package **service**, per la comunicazione con *API Gateway<sub>G</sub>*;
- Ampliare la lista presente in **CreateBlockActivity** (package **activities**) in modo che presenti un elemento cliccabile in più;
- Definire un widget a scelta (*activity* o *fragment*) che rappresenti l'inserimento dei dati da parte dell'utente;

- Inserire il listener dell'elemento aggiunto alla lista nel widget appena creato;
- Nel caso in cui sia stato creato un fragment ampliare la funzione *onFragmentClick(sender)* in modo che registri e notifichi alle *activity<sub>G</sub>* interessate le informazioni per l'aggiunta del blocco, in alternativa, se è stata creata una nuova activity, gestire l'interazione con essa mediante le funzioni *startActivityForResult(intent, context)* e *onActivityResult(requestCode, resultCode, data)*;
- Nella *CreateWorkflowActivity* e *ViewBlockActivity*, nelle funzioni *onActivityResult(requestCode, resultCode, data)* prelevare i dati passati dalle altre activity e chiamare il *viewModel* di conseguenza;
- Aggiornare il *viewModel* in modo che supporti l'aggiunta del blocco appena creato.

## 5.2 Estensione skill (lambda)

La Skill può essere estesa in questi modi:

- Aggiunta di un nuovo blocco §5.2.1;
- Aggiunta di nuove frasi per Alexa §5.2.2;
- Aggiunta di un nuovo metodo d'accesso al database §5.2.3;
- Aggiunta di un nuovo sistema di autenticazione §5.2.4.

### 5.2.1 Nuovo blocco

Un nuovo blocco può essere **semplice**, **filtrabile**, **filtrabile + elicito**, **elicit**. Per creare un nuovo blocco, bisogna seguire i seguenti passaggi:

- Creare una classe che rappresenta il blocco ed implementa *Block* (oppure implementa *Filterable* se il blocco è **filtrabile** o **filtrabile + elicito**), implementando i metodi rispettando le loro firme;
- Nella classe *workflow<sub>G</sub>*, all'attributo *createBlockCommands*, aggiungere una coppia chiave - funzione. La chiave è una stringa contenente il nome del blocco, la funzione è *Promise<Block> => Promise.resolve(new BlockClassName(config))*, dove *BlockClassName* è il nome della nuova classe;

- **Solo se il blocco è elicito oppure filterabile + elicito:** Creare la classe che rappresenta il connettore del blocco, implementando `Connector`. Questa classe deve occuparsi di tutte le connessioni con servizi esterni alla *skill<sub>G</sub>*.

### 5.2.2 Nuove frasi per Alexa

Il file `phrases-EN.json` e `phrases-IT.json` contengono tutte le frasi personalizzate che Alexa può dire. A ogni parola chiave (uguale per tutte le lingue) sono associate delle frasi (tradotte nella lingua relativa al file).

Per inserire una nuova frase, è necessario inserirla nel relativo oggetto associato alla parola chiave. E' importante inserire la frase tradotta per tutti i file `phrases-LINGUA.json`.

Per inserire una nuova parola chiave:

- Inserire un oggetto contenente come chiave la **parola chiave** e come contenuto una lista di stringhe, contenenti le frasi;
- Ripetere l'operazione precedente per ogni lingua;
- Inserire un nuovo metodo nella classe *PhrasesGenerator.ts*, che ritorna la frase che Alexa deve dire;
- L'attributo statico **jsonPhrases** contiene il file *phrases-LINGUA.json* (la selezione della lingua viene fatta in automatico);
- Chiamare questo metodo in modo statico dove è necessario.

Per inserire una nuova lingua:

- Inserire un nuovo file *phrases-LINGUA.json*, contenente tutte le parole chiavi presenti negli altri file di configurazione e le relative frasi tradotte;
- Nel metodo `setLanguage` aggiungere un case con la nuova lingua.

### 5.2.3 Nuovo metodo d'accesso al database

Tutti gli accessi al database, per quanto riguarda il download dei *workflow<sub>G</sub>*, avvengono attraverso la classe **WorkflowService**, nel metodo *workflowFromDatabase*.

É possibile aggiungere una nuova funzione in **WorkflowService**, contenente un nuovo metodo d'accesso a un diverso database. Per esempio, può essere



fatto attraverso una chiamata HTTP a un servizio REST, oppure usando un *web socket*<sub>G</sub>. È necessario che il valore ritornato sia una lista di blocchi. Lo schema del blocco si può trovare nella cartella *JSONconfigurations*, nel file *JSONconfiguration.ts*, con la chiave (**BlockJSON**).

### 5.2.4 Nuovo sistema di autenticazione

L'autenticazione, nella versione 1.0.0 della *skill*<sub>G</sub>, avviene solo attraverso Amazon.

Il file contenente la richiesta dell'utente (dal Alexa) contiene anche un **accessToken**. Questo viene utilizzato dalla classe **User**, dal metodo *credentialsByAccessToken*, per estrarre l'**AmazonID** dell'utente (lo stesso salvato nel database).

Questo sistema è basato su oauth, anche se tutta la parte a basso livello è gestita da Amazon.

Per aggiungere un altro sistema di autenticazione basato su oauth:

- creare una nuova configurazione Oauth e ottenere i dati d'accesso (URI, clientID, secret...);
- aprire il pannello di controllo di Alexa<sup>1</sup>;
- accedere alla skill MegAlexa;
- aprire la sezione "account linking";
- inserire i dati richiesti, generati da *OAuth*<sub>G</sub>;
- nella skill MegAlexa, modificare la funzione *credentialsByAccessToken*, nella classe **User**, così che sia compatibile con il nuovo servizio (ciò che cambia sarà l'ID e la chiamata GET per ottenerlo).

---

<sup>1</sup><https://developer.amazon.com/alexa/console/ask>

# Appendice A

## Glossario

### 7

#### **7z**

Software utilizzato per scompattare le cartelle compresse.

### A

#### **Activity**

Interfaccia utente mediante la quale si consultano dati o si immettono input.

#### **Adapter**

Pattern strutturale basato su classi ed oggetti che risolve il problema dell'interoperabilità tra interfacce differenti.

#### **Amazon**

Azienda di commercio elettronico con sede negli USA.

#### **Android**

Sistema operativo per dispositivi mobili sviluppato da Google Inc.

#### **Android Studio**

Software utilizzato per lo sviluppo di applicazioni Android.

## API

(Application Programming Interface) sono degli strumenti di programmazione che le software house mettono a disposizione degli sviluppatori per aiutarli nella creazione di applicazioni.

## API Gateway

Servizio di AWS per la creazione, manutenzione e protezione di API.

## AWS Lambda

Servizio serverless offerto da AWS che consente di eseguire codice in risposta ad eventi.

## C

### Connettore

Descrive un modulo la cui funzione è favorire la comunicazione e collaborazione fra altri moduli.

## D

### DynamoDB

Database non sequenziale messo a disposizione da AWS che può contenere dati di tipo documento e di tipo chiave-valore.

## E

### Echo

E' un altoparlante intelligente sviluppato da Amazon controllato tramite comandi vocali, senza dover utilizzare le mani, che si connette ad Alexa per riprodurre musica, rispondere a domande, leggere le ultime notizie, fornire le previsioni del tempo e molto altro.

## G

### Git

Sistema di controllo di versione utilizzato da GitHub.

**Gradle**

Sistema open source per l'automazione dello sviluppo.

**J****JSON**

(JavaScript Object Notation) Formato adatto all'interscambio di dati fra applicazioni client/server.

**K****Kotlin**

Linguaggio di programmazione per sistemi Android

**M****Model-View-ViewModel (MVVM)**

Design pattern architetturale di progettazione software.

**O****OAuth**

Protocollo di rete utilizzato per lavorare con HTTP.

**S****Skill**

Per skill si intende un applicativo sviluppato su Amazon Alexa in grado di eseguire un compito specifico.

**T****TypeScript**

Linguaggio di programmazione sviluppato da Microsoft, la cui compilazione genera codice JavaScript standard

# W

## **Workflow**

Consiste in una serie di connettori eseguiti da Alexa in successione uno dopo l'altro secondo le esigenze dell'utente.