# MegAlexa

Arricchitore di skill di Amazon Alexa

---

# Manuale dello Sviluppatore

Gruppo ZeroSeven


| | |
|---|---|
| **Versione** | 0.0.5 |
| **Data Redazione** | 2019-03-29 |
| **Redazione** | Mirko Franco |
| | Stefano Zanatta |
| **Verifica** | Matteo Depascale |
| **Approvazione** | Stefano Zanatta |
| **Uso** | Esterno |
| **Distribuzione** | Prof. Tullio Vardanega |
| | Prof. Riccardo Cardin |
| | Gruppo ZeroSeven |
| | Zero12 s.r.l. |
| **Email di contatto** | zerosevenswe@gmail.com |

## Registro delle modifiche

| Versione | Data | Descrizione | Autore | Ruolo |
|---|---|---|---|---|
| 0.0.7 | 2019-04-09 | Stesura §6, §5.1 | Stefano Zanatta | Progettista |
| 0.0.6 | 2019-04-08 | Stesura §4.2.1, §3.1 | Stefano Zanatta | Progettista |
| 0.0.5 | 2019-04-05 | Stesura capitolo §4 | Mirko Franco | Progettista |
| 0.0.4 | 2019-04-05 | Stesura capitolo §2 | 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 | Creato documento | Mirko Franco | Progettista |

# Indice

# Elenco delle figure

# Elenco delle tabelle

# 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 $Alexa_G$ di $Amazon_G$. L'obiettivo è la creazione di una $skill_G$ in grado di avviare $workflow_G$ creati dagli utenti fornendogli dei $connettori_G$.

## 1.3 Glossario

Al fine di evitare ogni ambiguità di linguaggio e massimizzare la comprensione dei documenti, i termini tecnici, di dominio, gli acronimi e le parole che necessitano di essere chiarite, sono riportate nel *Glossario v3.0.0*. Ogni occorrenza di vocaboli presenti nel Glossario è marcata da una "G" maiuscola in pedice.

# Capitolo 2

# Requisiti di Sistema

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

## 2.1 Applicazione

L'applicazione è compatibile con tutti i dispositivi Android con versione 4.4 o superiore.

## 2.2 Skill

La skill è compatibile con tutti i dispositivi Amazon Echo.

# Capitolo 3

# Procedura di installazione

## 3.1 Installazione Skill

Per ognuno dei sequenti 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 7z e inserire il suo eseguibile tra le variabili di sistema;
- installare e configurare aws-cli[1];
- 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.*

[1] https://aws.amazon.com/it/cli/

# Capitolo 4

# Tecnologie utilizzate

## 4.1 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 businesses 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.

### 4.1.1 AWS DynamoDB

Amazon DynamoDB 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.

### 4.1.2 AWS Lambda

AWS Lambda 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. É 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.

### 4.1.3 AWS API Gateway

AWS API Gateway 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 simultanee, inclusi gestione del traffico, controllo di accessi e autorizzazioni, monitoraggio e gestione delle versione 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.

### 4.1.4 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.

### 4.1.5 Node.js

La skill è sviluppata attraverso un progetto Node js. Tutte le classi (apparte index.js) sono scritte in Typescript, ma vengono compilate in Javascript prima di essere trasferite in AWS Lambda.

### 4.1.6 Npm

Gestore di package per node.js. Permette l'esecuzione di comandi personalizzati (come descritto in §3.1).

### 4.1.7 7z

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

## 4.2 Dipendenze esterne

### 4.2.1 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;
- **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;
- **@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.

### 4.2.2 App Android

# Capitolo 5

# Architettura

## 5.1 Skill

La skill è hostata nel sito https://developer.amazon.com/alexa/ (è necessario un account developer Amazon per accedervi). La skill è rappresentata da un JSON, configurabile attraverso l'UI fornito dalla piattaforma Alexa developer.

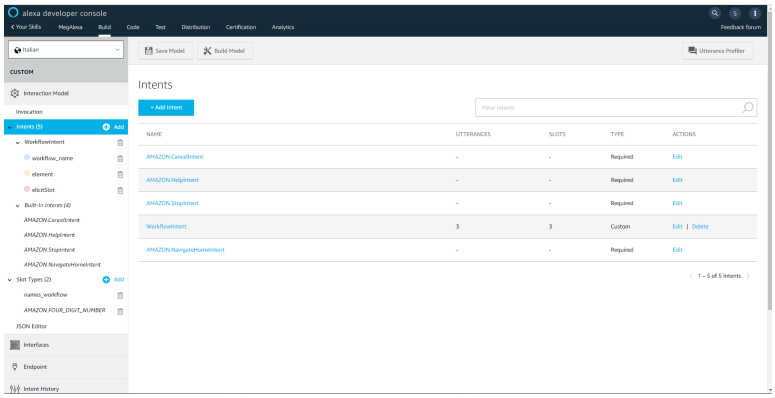


**Figura 5.1:** *Alexa developer platform*

La logica della skill è hostata in una AWS Lambda, quindi è serverless. Questo significa che viene creata una nuova istanza della Skill ogni volta che arriva una richiesta da Alexa (ogni utente genere 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[1].
Le informazioni più importanti contenute nel file JSON 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 viene hostata. 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'altro della skill (lambda).

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

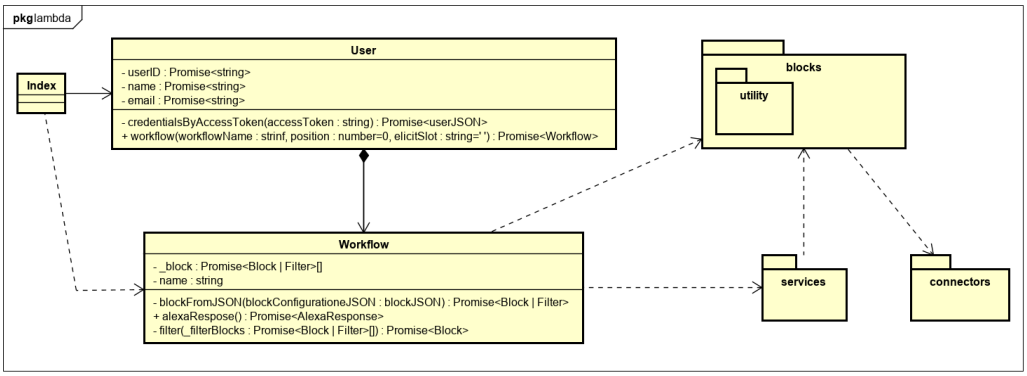


**Figura 5.2:** *Skill class diagram*

[1] 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. L'index è scritto in Javascript (per compatibilità con ask-sdk). Prima di fare il deploy in AWS Lambda, i file TypeScript vengono compilati in Javascript.

### 5.1.1 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 §5.1. Se nessun handler può gestire la richiesta, viene invocato Error handler;
- handle: se canHandle ritorna True, questa è la funzione che viene eseguita. deve ritornare una risposta comprensibile da Alexa.

### 5.1.2 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, permettendo di autenticare l'utente nel database;
- **Workflow:** usa WorkflowService per creare un Workflow a partire dalla sua rappresentazione in formato JSON.

### 5.1.3 Classe Workflow

In *Workflow* sono presenti i seguenti metodi:

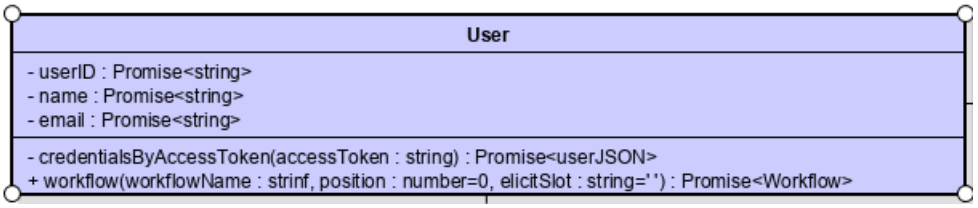


**Figura 5.3:** *Skill class diagram - User*

- **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.

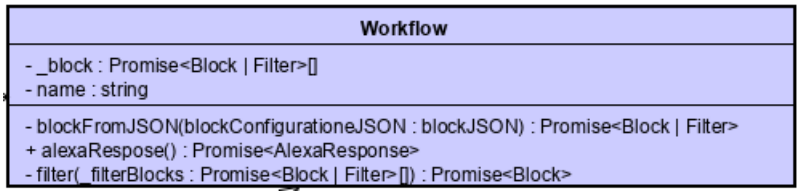


**Figura 5.4:** *Skill class diagram - Workflow*

### 5.1.4 Package services

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

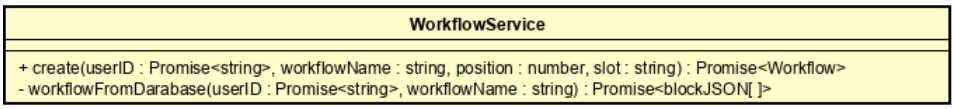


**Figura 5.5:** *Skill class diagram - WorkflowService*

### 5.1.5 Package blocks

Il package blocks contiene tutti i blocchi della skill 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 eseguire.

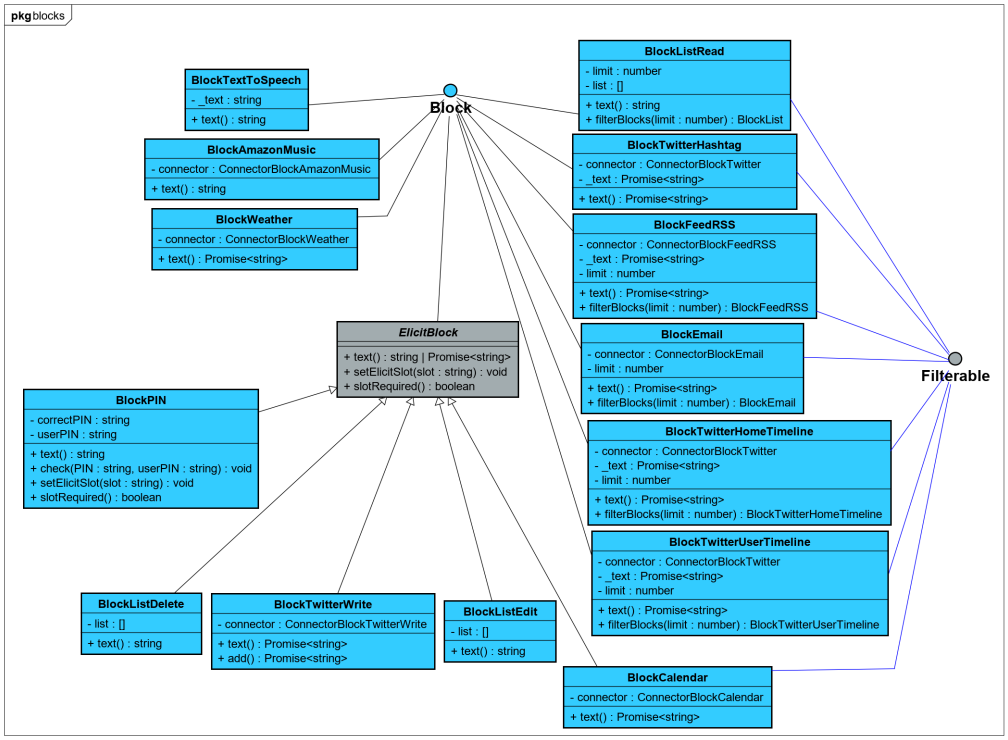


**Figura 5.6:** *Skill class diagram - blocks package*

#### 5.1.5.1 Package blocks.utils

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

### 5.1.6 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 liberia per conoscere il meteo di una certa zona.
Ogni Connector deve processare il risultato e trasformarlo nel testo che Alexa dovrà ripetere.

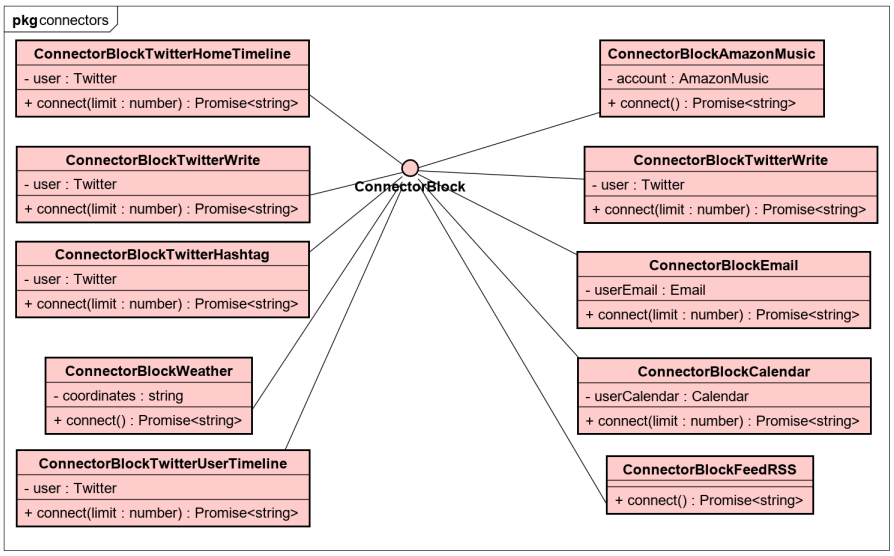


**Figura 5.7:** *Skill class diagram - blocks package*

# Capitolo 6

# Estensione delle funzionalità

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

## 6.1 Estensione Skill (lambda)

La Skill può essere estesa in questi modi:

- aggiunta di un nuovo blocco§6.1.1;
- aggiunta di nuove frasi per Alexa§6.1.2;
- aggiunta di un nuovo metodo d'accesso al database§6.1.3;
- aggiunta di un nuovo sistema di autenticazione§6.1.4;

### 6.1.1 Nuovo blocco

Questa sezione verrà redatta quando il prodotto sarà completo.

### 6.1.2 Nuove frasi per Alexa

Questa sezione verrà redatta quando il prodotto sarà completo.

### 6.1.3 Nuovo metodo d'accesso al database

Questa sezione verrà redatta quando il prodotto sarà completo.

### 6.1.4 Nuovo sistema di autenticazione

Questa sezione verrà redatta quando il prodotto sarà completo.