HEXADEC

Manuale dello Sviluppatore
Progetto MegAlexa

hexadec.swe@gmail.com

Informazioni sul documento

Versione	1.0.0
Responsabile	Andrea Chinello
Redattori	Giacomo Corrò, Sukhjinder Singh, Francesco Barbanti
Verificatori	Daniele Scialabba, Davide Tognon, Valentin Grigoras
Uso	Esterno
Destinatari	Prof. Tullio Vardanega, Prof. Riccardo Cardin, zero12

Registro delle Modifiche

Versione	Data	Autore	Ruolo	Descrizione
1.0.0	2019-07-10	Andrea Chinello	Responsabile	Approvazione documento per rilascio
0.5.0	2019-07-09	Valentin Grigoras	Verificatore	Verifica documento
0.4.2	2019-07-08	Giacomo Corrò	Redattore	Modifiche in §5.6.3
0.4.1	2019-07-08	Francesco Barbanti	Redattore	Modifiche in §6.3.1
0.4.0	2019-07-07	Daniele Scialabba	Verificatore	Verifica documento
0.3.6	2019-07-06	Sukhjinder Singh	Redattore	Aggiunta §5.7.1.2 e §5.7.2.2
0.3.5	2019-07-05	Francesco Barbanti	Redattore	Aggiunta §6.3
0.3.4	2019-07-04	Giacomo Corrò	Redattore	Aggiunta §5.7
0.3.3	2019-07-01	Sukhjinder Singh	Redattore	Ampliate descrizioni UML in §5.6.2, §5.6.3 e §5.6.4
0.3.2	2019-06-28	Giacomo Corrò	Redattore	Ridotta descrizione in §4.2
0.3.1	2019-06-27	Giacomo Corrò	Redattore	Correzione immagini in §5.2 e §6.1
0.3.0	2019-06-09	Daniele Scialabba	Verificatore	Verifica documento

Versione	Data	Autore	Ruolo	Descrizione
0.2.1	2019-06-08	Valentin Grigoras	Redattore	Stesura §9
0.2.0	2019-06-08	Daniele Scialabba	Verificatore	Verifica documento
0.1.2	2019-06-07	Sukhjinder Singh	Redattore	Correzione errori in §6.5 e §7.3.2
0.1.1	2019-06-07	Sukhjinder Singh	Redattore	Stesura §A
0.1.0	2019-06-07	Andrea Chinello	Verificatore	Verifica documento
0.0.8	2019-06-05	Francesco Barbanti	Redattore	Stesura §8
0.0.7	2019-06-04	Valentin Grigoras	Redattore	Stesura §7.1 e §7.2
0.0.6	2019-06-02	Francesco Barbanti	Redattore	Stesura di §7.3
0.0.5	2019-06-01	Valentin Grigoras	Redattore	Stesura di §5
0.0.4	2019-05-30	Sukhjinder Singh	Redattore	Stesura di §4.2 e §6
0.0.3	2019-05-30	Valentin Grigoras	Redattore	Stesura di §3 e §4.1
0.0.2	2019-05-29	Valentin Grigoras	Redattore	Stesura di §1 e §2
0.0.1	2019-05-23	Davide Tognon	Responsabile	Creazione del template

Contenuti

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Il presente documento ha lo scopo di definire nel dettaglio le componenti dell'architettura software HexadecApp e di illustrare al manutentore del software le modalità di installazione. Descrive inoltre, i requisiti necessari per poterlo utilizzare, le librerie e i *framework*G esterni adottati per lo sviluppo dell'applicazione.

1.2 Scopo del prodotto

Lo scopo del prodotto è sviluppare una *skill*G per l'assistente vocale Amazon Alexa in grado di avviare dei *workflow*G personalizzati per mezzo di un' applicazione Android. Il prodotto comprenderà:

- un'applicazione Android realizzata in *Kotlin*G che permetterà all'utente utilizzatore di creare e gestire i workflow;
- la skill MegAlexa realizzata con i servizi di Amazon tra cui *Lambda*G, *DynamoDB*G, e *API Gateway*G.

1.3 Note esplicative

Al fine di evitare ambiguità ai lettori non interni al gruppo, si specifica l'utilizzo di convenzioni prese da HexaDec per la stesura dei documenti, che sono le seguenti:

- **Glossario**: per evitare ridondanze e ambiguità di linguaggio e massimizzare la comprensione dei documenti, i termini tecnici, di dominio, e gli acronimi che necessitano di una spiegazione, sono definiti e descritti nel *Glossario v4.0.0*D. I vocaboli riportati nel *Glossario v4.0.0*D sono marcati da una "G" maiuscola a pedice;
- **Documentazione**: i nomi degli altri documenti prodotti dal gruppo HexaDec compariranno sempre in corsivo, seguiti da una "D" a pedice.

1.4 Riferimenti

1.4.1 Riferimenti informativi

- Materiale del corso di Ingegneria del Software:
  - diagrammi delle classi:
    https://www.math.unipd.it/~tullio/IS-1/2018/Dispense/E03b.pdf;
  - diagrammi dei package:
    https://www.math.unipd.it/~tullio/IS-1/2018/Dispense/E05a.pdf;
  - diagrammi di sequenza:
    https://www.math.unipd.it/~tullio/IS-1/2018/Dispense/E04a.pdf.
- Documentazione del linguaggio di programmazione Kotlin:
  https://www.kotlinlang.org/docs/reference.
- Documentazione della piattaforma Node.js:
  https://nodejs.org/api.
- Documentazione dei servizi AWS:

- AWS Lambda:
  https://aws.amazon.com/it/lambda;
- AWS DynamoDB:
  https://aws.amazon.com/it/dynamodb;
- AWS Api Gateway:
  https://aws.amazon.com/it/api-gateway.

2 Tecnologie utilizzate

2.1 Front-end

2.1.1 Kotlin

Kotlin è un linguaggio di programmazione *general purpose*G, *multi-paradigma*G, *open source*G sviluppato dall'azienda di software *JetBrains*G.
L'obiettivo primario di Kotlin è mettere a disposizione dello sviluppatore un linguaggio di programmazione conciso e produttivo che sia completamente interoperabile con Java in entrambe le direzioni: Kotlin permette di utilizzare senza alcuna complicazione librerie Java pre-esistenti e, a loro volta, le applicazioni scritte in Kotlin sono invocabili da Java.
I motivi che ci hanno spinto a utilizzare questo linguaggio sono i seguenti:

- sintassi semplice;
- sintassi più leggibile rispetto a Java;
- il riconoscimento ufficiale da parte di Google come linguaggio di riferimento per lo sviluppo mobile per la piattaforma Android.

2.1.2 XML

XML è un linguaggio di marcatura estensibile sviluppato e controllato dal *World Wide Web Consortium*G. Serve a creare linguaggi di markup che descrivono i dati di qualsiasi tipo e in modo strutturato. Android fornisce gli schemi necessari per comporre widget e posizionarli sullo schermo, separando ulteriormente la parte logica da quella grafica.

2.2 Back-end

2.2.1 Amazon Web Service

Amazon Web Services offre un'ampia gamma di servizi globali basati sul cloud per elaborazione, storage, database, e altre funzionalità per aiutare il business ad essere scalabile e crescere con facilità. AWS fornisce infatti prodotti e soluzioni per creare applicazioni sofisticate, caratterizzate da una maggiore flessibilità, scalabilità e affidabilità.

2.2.1.1 AWS DynamoDB

Come *DBMS*G per l'applicativo mobile HexadecApp si è utilizzato DynamoDB. Amazon DynamoDB è un servizio di database NoSQL interamente gestito che combina prestazioni elevate e prevedibili con una scalabilità ottimale. Supporta i modelli di dati di tipo documento e di tipo chiave-valore che offre prestazioni di pochi millisecondi a qualsiasi livello.

2.2.1.2 AWS Lambda

*AWS Lambda*G è un servizio che permette di ospitare il codice e di farlo eseguire quando certi eventi si verificano. Una volta caricato il codice, Lambda si prende carico delle azioni necessarie per eseguirlo e ricalibrarne le risorse con massima disponibilità. Il codice può essere configurato in modo che venga attivato automaticamente da altri servizi AWS oppure in modo che venga richiamato direttamente da un qualsiasi applicazione mobile o web.

2.2.1.3 AWS API Gateway

Amazon API Gateway è un servizio AWS che consente di creare, pubblicare, gestire, monitorare e proteggere le proprie API REST su qualsiasi scala. Questo servizio offre la possibilità di creare API sicure e scalabili che accedono ad AWS o ad altri servizi Web e ai dati archiviati nel cloud AWS. Si possono inoltre creare API da utilizzare nelle applicazioni client (app) oppure puoi rendere le tue API disponibili per gli sviluppatori di applicazioni di terze parti. API Gateway non prevede alcuna tariffa minima nè costi di attivazione; saranno addebitati solamente le chiamate API ricevute e i volumi di dati trasferiti in uscita.

2.2.2 Node.js

Node.js è un framework, che viene usata per scrivere applicazioni in Javascript lato server. Abbiamo sviluppato la skill utilizzando Node.js. Tutte le classi sono scritte in TypeScript, ma vengono compilate in JavaScript prima di essere caricate sulla console AWS Lambda.

2.2.3 TypeScript

TypeScript è un *linguaggio di programmazione libero*G ed open source sviluppato da Microsoft. Estende la sintassi di JavaScript e in questo modo qualunque programma scritto in JavaScript è anche in grado di funzionare con TypeScript senza nessuna modifica. Non essendo Javascript un linguaggio tipizzato, ai fini di ciò che serviva per la progettazione del back-end della skill, è stato utilizzato TypeScript. Esso estende Javascript aggiungendo varie funzionalità, in particolare:

- **interfacce**;
- **classi**;
- **moduli**;
- **tipi di dato**, anche se opzionali, per gli obiettivi di progettazione sarà fondamentale questa caratteristica.

2.2.4 Npm

Npm è un gestore di pacchetti per Node.js che offre centinaia di migliaia di packages. L'obiettivo principale è la dipendenza automatica e la gestione dei packages. Ciò significa che è possibile specificare tutte le dipendenze del progetto all'interno del file package.json, quindi chiunque avesse bisogno di svolgere operazioni sulla skill, necessita di eseguire l'installazione di npm in modo da disporre immediatamente di tutte le dipendenze installate. L'installazione va effettuata attraverso il comando *npm install*. Inoltre è possibile specificare da quali versioni dipende la skill per evitare che gli aggiornamenti non siano compatibili con la versione utilizzata.

2.2.5 7z

Per poter caricare i file del codice sorgente della skill è richiesto che essi siano compressi. Il gruppo utilizza 7z per comprimere i file.

3 Prerequisiti

3.1 Requisiti software

- Java Development Kit (jdk) 1.8. o superiore;
- Android Studio 3.3 o superiore.

3.2 Requisiti minimi di sistema

- Windows 7/8/10 (32 o 64 bit) oppure Ubuntu 12.04 o superiore (32 o 64 bit);
- 3 GB RAM;
- 2 GB di spazio libero sul disco;
- 1280x800 px di risoluzione schermo;
- connessione ad internet per scaricare l'applicativo;
- un qualsiasi browser internet .

3.3 Requisiti di sistema consigliati dal gruppo HexaDec

- Windows 7/8/10 (32 o 64 bit) oppure Ubuntu 12.04 o superiore (32 o 64 bit);
- 8 GB RAM + 2 GB per l'emulatore Android per un totale di 10GB;
- 4 GB di spazio libero sul disco;
- 1280x800 px di risoluzione schermo;
- connessione ad internet per scaricare l'applicativo;
- il browser Mozilla Firefox o Google Chrome nelle più recenti versioni.

3.4 Account Amazon

È necessario avere un account Amazon per utilizzare HexadecApp; se non si è già in possesso di un account lo si può creare in 2 modi:

1. Tramite l'applicazione HexadecApp:
    - premere sul pulsante "login" :

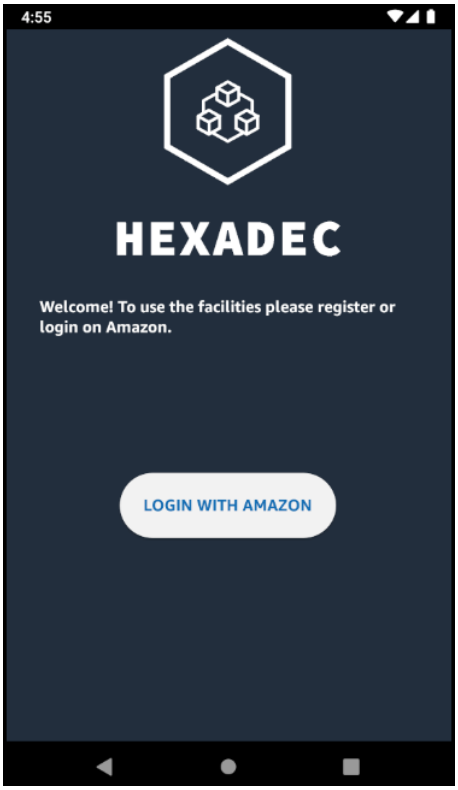


Figura 1: Pagina principale HexadecApp

- premere sul pulsante ”create a new Amazon account” :

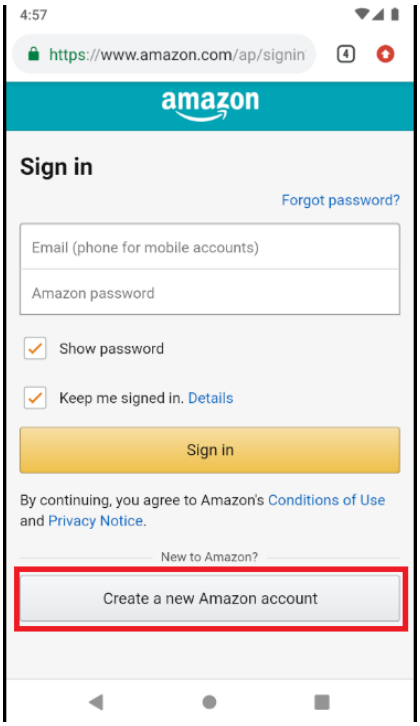


Figura 2: Pagina login Amazon

- compila tutti i campi e completa la registrazione :

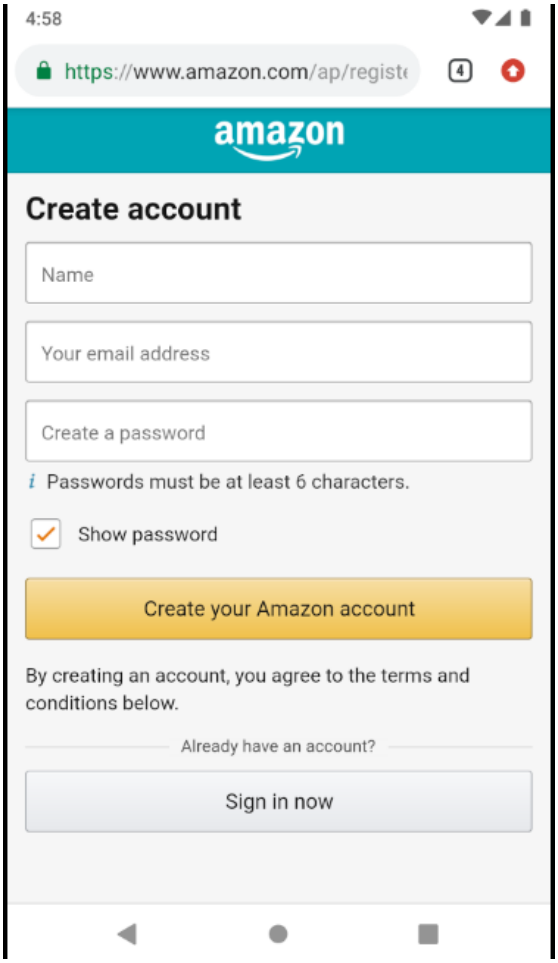


Figura 3: Pagina registrazione Amazon

2. Direttamente dalla pagina di registrazione di Amazon raggiungibile a questo indirizzo:
   https://www.amazon.com/ap/register.

3.5 Node.js con Node Package Manager (npm) versione 6.9.0

Npm è un gestore di pacchetti per il linguaggio di programmazione JavaScript. È il gestore di pacchetti predefinito per l'ambiente di runtime JavaScript Node.js. È possibile eseguire l'installazione seguendo le istruzioni su:

https://nodejs.org/it/download

Nota Bene: La §3.6 descrive l'installazione di Node.js con il pacchetto Node Package Manager (npm) incluso. Nel caso in cui si volesse installare solo Node.js consultare la guida presente al seguente link: https://nodejs.org/en/download/package-manager/#windows

4 Installazione

4.1 Clonazione repository

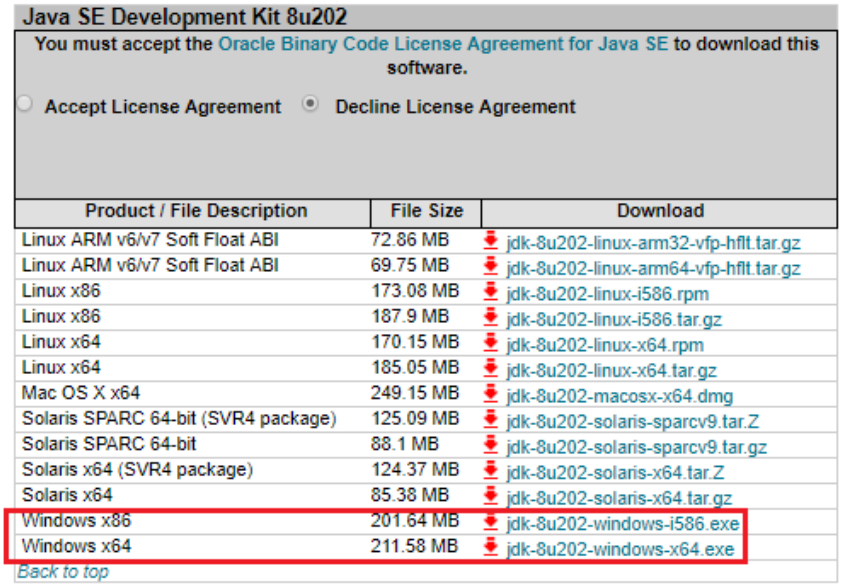
Per clonare la repository sulla propria macchina, aprire il terminale e digitale il seguente commando:

- **Skill**: *git clone https://github.com/HexadecSwe16/HexadecAppSkill*;
- **App**: *git clone https://github.com/HexadecSwe16/HexadecAppAndroid*.

4.2 Android Studio

1. assicurarsi di aver soddisfatto i requisiti minimi descritti nella §3.2;
2. recarsi al sito Oracle dal seguente indirizzo:

   https://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase8-2177648.html
3. scaricare il pacchetto corretto in base alla propria versione di Windows:



Java SE Development Kit 8u202

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement Decline License Agreement

Product / File Description	File Size	Download
Linux ARM v6/v7 Soft Float ABI	72.86 MB	jdk-8u202-linux-arm32-vfp-hflt.tar.gz
Linux ARM v6/v7 Soft Float ABI	69.75 MB	jdk-8u202-linux-arm64-vfp-hflt.tar.gz
Linux x86	173.08 MB	jdk-8u202-linux-i586.rpm
Linux x86	187.9 MB	jdk-8u202-linux-i586.tar.gz
Linux x64	170.15 MB	jdk-8u202-linux-x64.rpm
Linux x64	185.05 MB	jdk-8u202-linux-x64.tar.gz
Mac OS X x64	249.15 MB	jdk-8u202-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	125.09 MB	jdk-8u202-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	88.1 MB	jdk-8u202-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	124.37 MB	jdk-8u202-solaris-x64.tar.Z
Solaris x64	85.38 MB	jdk-8u202-solaris-x64.tar.gz
Windows x86	201.64 MB	jdk-8u202-windows-i586.exe
Windows x64	211.58 MB	jdk-8u202-windows-x64.exe

Back to top

Figura 4: Java SE 8

4. avviare il download ed una volta terminato installare il pacchetto seguendo le istruzioni a schermo;
5. recarsi al seguente indirizzo e scaricare l'ultima versione di Android Studio disponibile:

   https://developer.android.com/studio
6. cliccare sul link "DOWNLOAD ANDROID STUDIO" mostrato in figura:

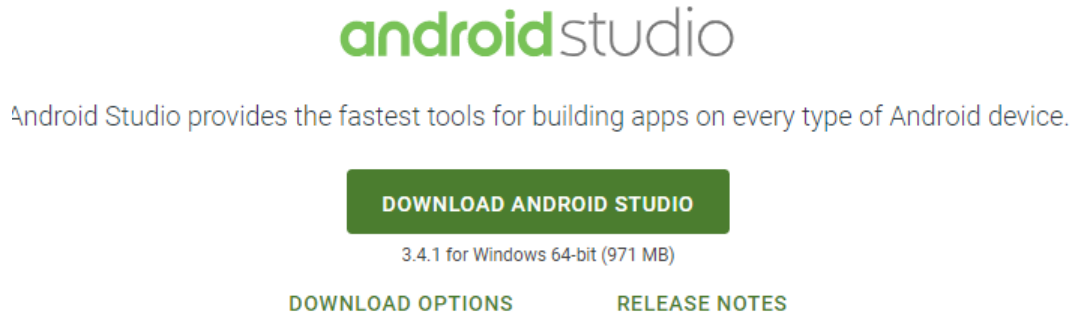


Figura 5: Download Android Studio

7. avviare il download ed una volta terminato cliccare sul corrispondente file eseguibile:

Figura 6: Installazione Android Studio

4.3 Skill

Questa guida descrive l'uso di ASK SDK v2 per Node.js.

4.3.1 Prerequisiti

- Un progetto NPM. Il seguente link è una guida al creazione di un progetto NPM. https://ask-sdk-for-nodejs.readthedocs.io/en/latest/Developing-Your-First-Skill.html;
- Un ambiente di sviluppo Node.js adatto. ASK SDK v2 per Node.js richiede Node 4.3.2 o una versione superiore.

4.3.2 Integrare ASK SDK al progetto

Per usare ASK SDK v2 per Node.js è necessario installarlo come un modulo NPM. Per fare ciò eseguire il seguente comando:

npm install –save ask-sdk

4.3.3 Sviluppo skill

4.3.3.1 Prerequisiti

- Un Amazon developer account per configurare la skill. È possibile crearne uno al seguente link: https://developer.amazon.com/;
- Un account AWS per ospitare la skill su AWS Lambda.
- Un progetto NPM, come descritto nella sezione precedente.

4.3.3.2 Creazione del pacchetto della skill

Una volta apportate le modifiche, è possibile creare il pacchetto della skill. Per preparare al caricamento su AWS Lambda, creare un file zip contenente tutti i file della skill.

4.3.3.3 Caricamento della skill su AWS Lambda

Caricare il file .zip creato al passo precedente sulla funzione Lambda collegata alla skill.

5 Architettura applicazione Android

5.1 Metodo e formalismo

L'architettura dell'applicazione verrà presentata seguendo un approccio *top-down*G, quindi descrivendo l'architettura dal generale allo specifico. Verranno descritti per primi i vari package con i relativi componenti e poi nel dettaglio le classi in esse contenute fornendo il diagramma UML con le relative dipendenze.

5.2 Architettura MVP

L'applicazione è stata sviluppata secondo il pattern *MVP (Model View Presenter)*G. Nello sviluppo Android questo è uno dei pattern più efficaci in quanto permette di separare la vista dalla logica dei dati riducendo il numero di dipendenze dirette tra il model e la view. Di seguito viene mostrato il diagramma dei package con le relative dipendenze:

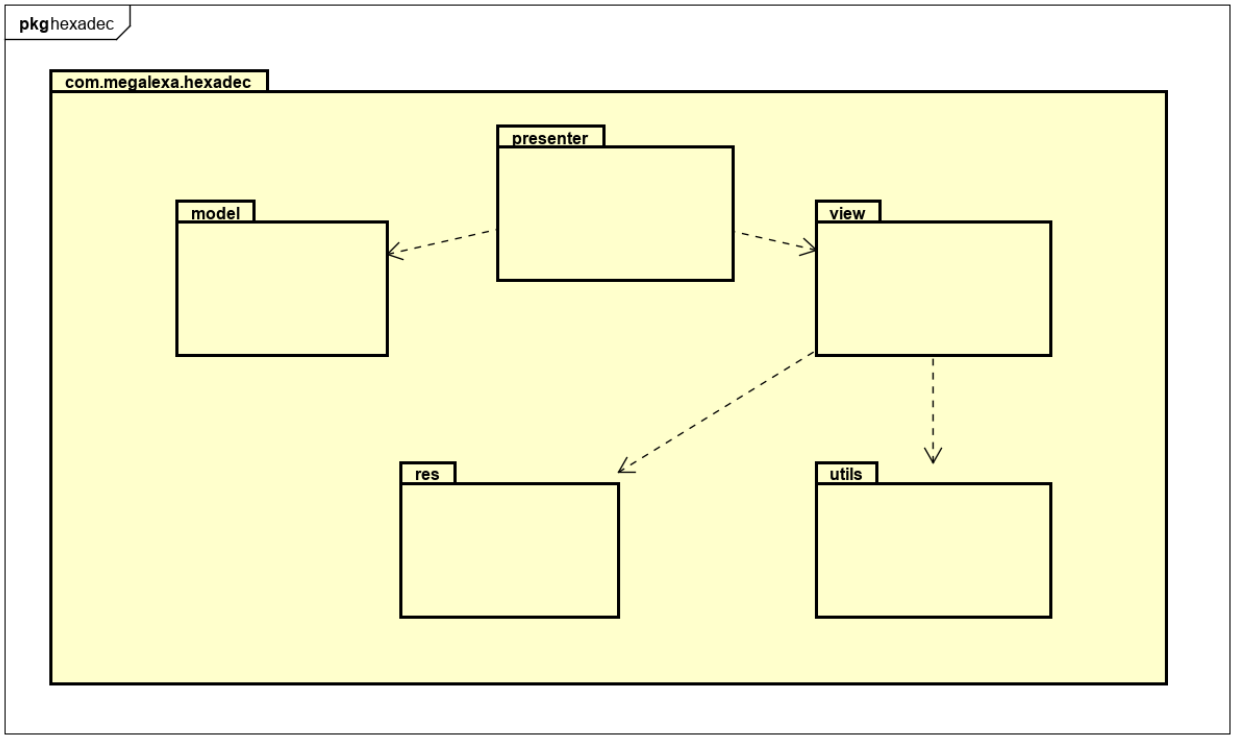


Figura 7: Diagramma dei package: MVP

Di seguito vengono descritti i vari package:

- **model**: contiene tutte le classi che fanno parte del modello;
- **presenter**: è il mediatore tra la vista e il modello, recupera i dati dal modello e li invia alle viste decidendo anche cosa succede quando si interagisce con la vista;
- **view**: definisce l'interfaccia utente ed è implementata tramite *Activity*G;
- **res**: contiene i file xml per definire interfaccia delle activity e i file strings utili per l'implementazione della funzionalità multi lingua;
- **utils**: contiene delle classi utili per definire gli elementi del contenitore *RecylerView*G.

5.3 Contratti

Abbiamo definito dei *contratti*[6] tramite l'uso di interfacce che definiscono la relazione tra view e presenter. Le tre interfacce fondamentali sono le seguenti:

```
interface BasePresenter {
    fun onDestroy()
}
```

Figura 8: Interfaccia: BasePresenter

- è un'interfaccia generica che dovrebbe essere implementata da qualsiasi presenter aggiunto al progetto.

```
interface BaseView<T> {
    fun setPresenter(presenter:T)
}
```

Figura 9: Interfaccia: BaseView

- simile al BasePresenter questa interfaccia dovrà essere implementata da tutte le view che creiamo. Siccome tutte le view sono legate ad un presenter, abbiamo deciso di templetizzare l'interfaccia BaseView in quanto le view devono settare un metodo *SetPresenter()*.

```
interface MainContract {
    interface LoginContract:BasePresenter{
        fun updateUser(userId:String,userName:String,userEmail:String)
        fun setUpWorkflow()
    }
    interface  LoginView:BaseView<LoginContract>{
    }
```

Figura 10: Interfaccia: MainContract

- questa interfaccia contiene tutte le dichiarazioni dei contratti tra le view e presenter. Il tipo T viene sostituito dal nome del presenter con il quale leghiamo la vista. Nell'interfaccia presenter vengono dichiarati tutti quei metodi che si occuperanno dell'aggiornamento della vista:

5.4 Modello

Qualsiasi applicazione che abbia un minimo di struttura manterrà la logica dei dati separata dall' interfaccia. La seguente immagine descrive il diagramma delle classi del modello:

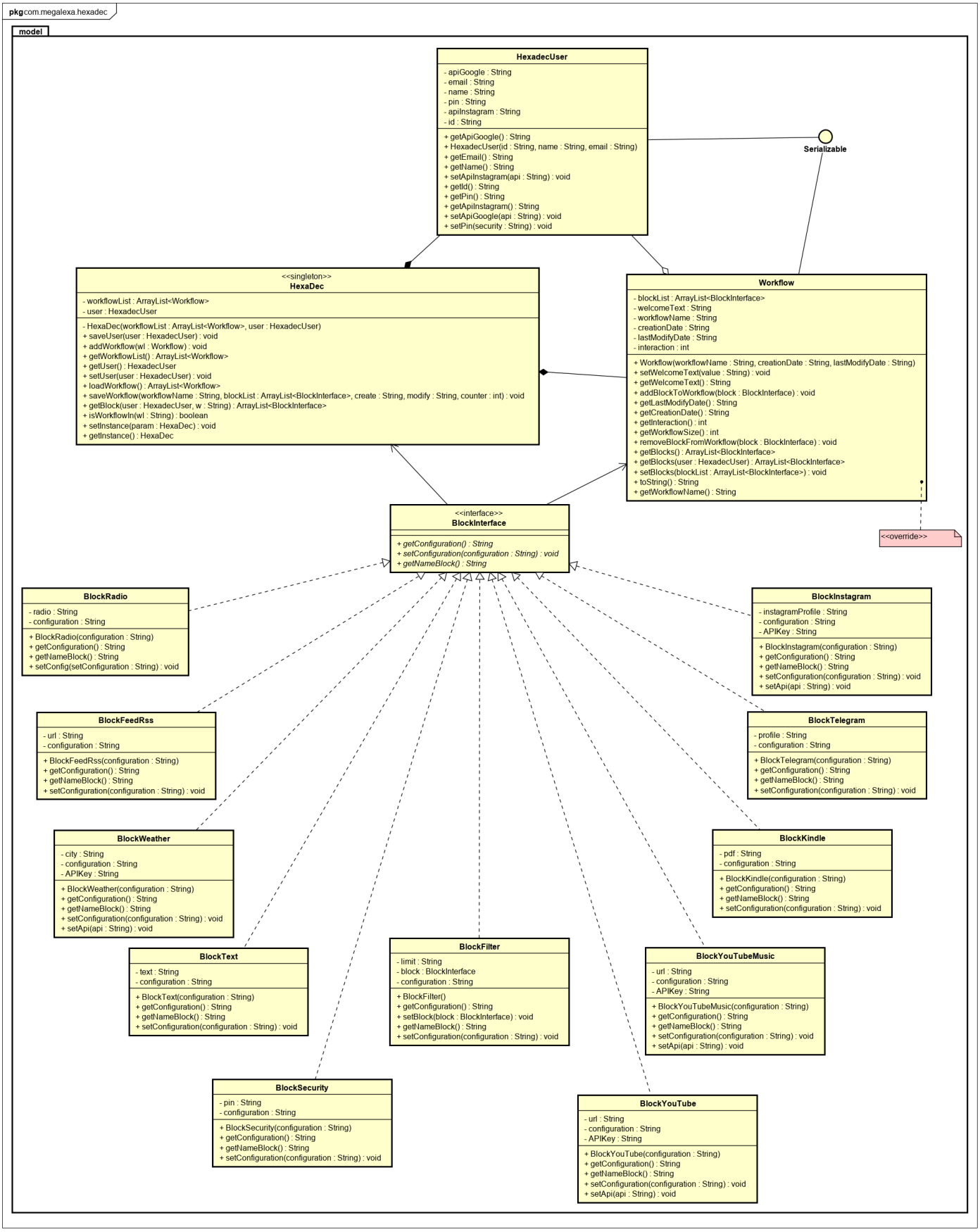


Figura 11: Diagramma delle classi: Modello

5.5 Presenter

Il presenter agisce da intermediario tra vista e modello. Recupera i dati dal modello e gli restituisce formattati al modello. Di seguito viene mostrato il diagramma delle classi dei presenter con le relative dipendenze:

La classe più rilevante è la HexaDec, in quanto verrà istanziata una sola volta dopo l'autenticazione, usando il pattern*singleton*G, in modo da evitare l'esistenza di due account attivi contemporaneamente sullo stesso dispositivo. HexaDec ha al suo interno un'istanza HexadecUser avente come campi, il nome dell'utente, il PIN (da usare nel blocco sicurezza) e la mail.

La classe HexaDec e' ulteriormente composta da una lista di Workflow che vengono richiesti alla basi di dati dopo aver effettuato l'accesso in modo da avere sempre aggiornati i workflow creati dall'utente attivo in quel momento.

5.6 View

Una view è l'interfaccia utente che appare sullo schermo. Il presenter agisce da intermediario tra vista e modello. Recupera i dati dal modello e li restituisce formattati al modello. Di seguito viene mostrato il diagramma delle classi dei presenter con le relative dipendenze:

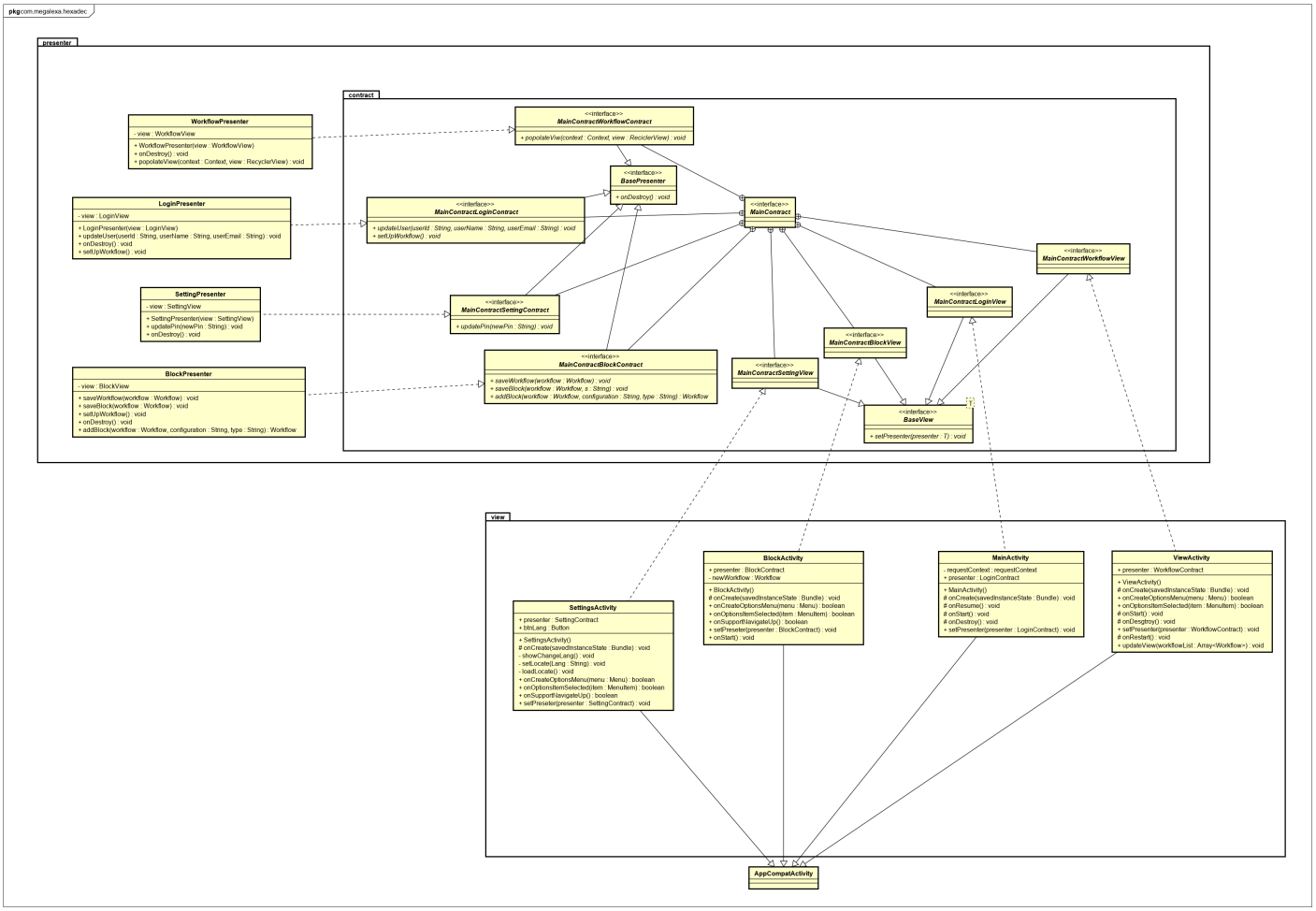


Figura 12: Diagramma delle classi: Views con i relativi presenter

5.6.1 MainViewActivity

E' la prima activity che viene aperta dall'applicazione; permette di fare il login, la registrazione e verificare se l'utente è autenticato o meno.

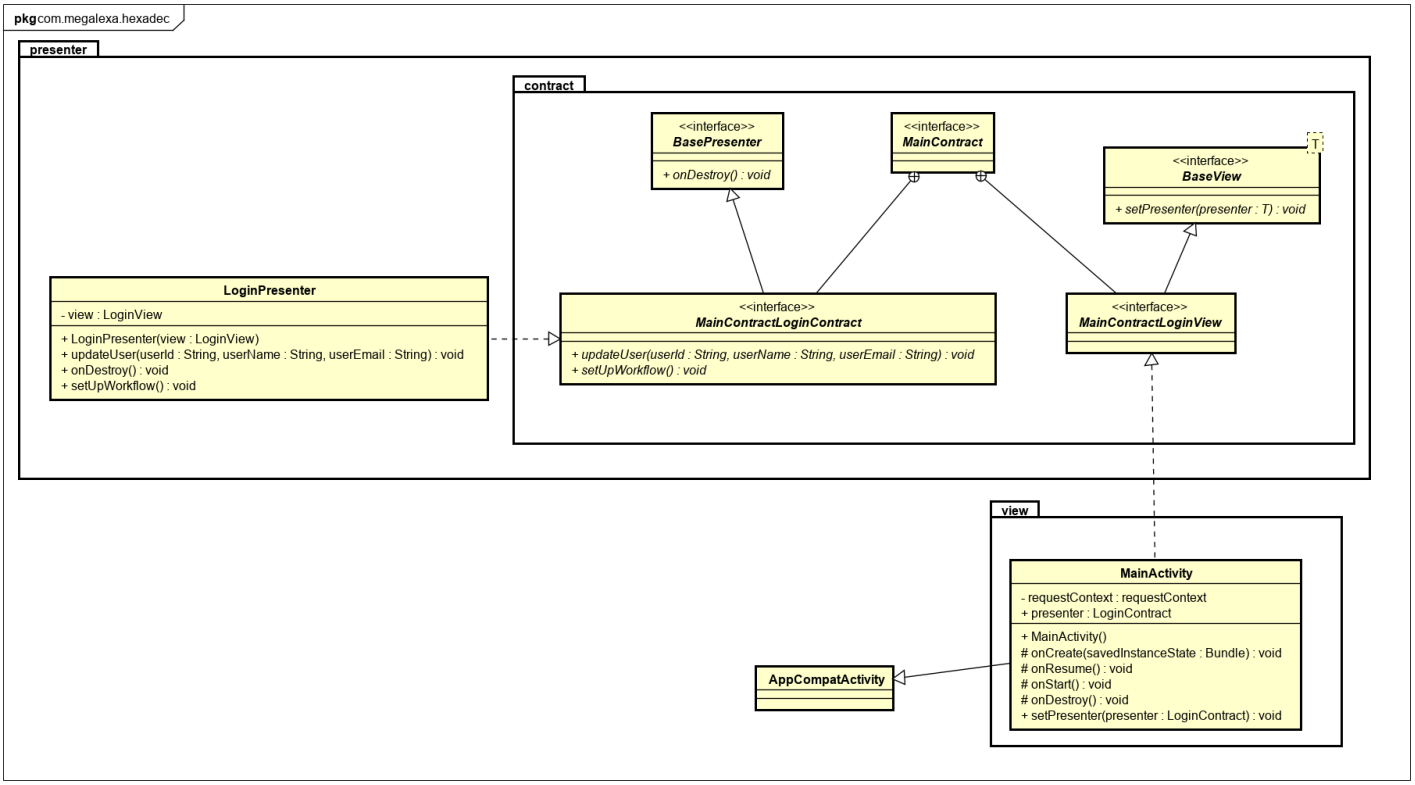


Figura 13: Diagramma delle classi: MainViewActivity

5.6.2 ViewActivity

E' la pagina principale dell'applicazione HexadecApp. Mostra la lista dei workflow dell'utente. Il recupero dei workflow avviene recuperando l'istanza dell'oggetto Singleton e iterando la lista *WorkflowList* viene popolata una recyclerView con gli worfklow dell'utente. Il popolamento viene implementato nel presenter ed il metodo che si occupa ti tutto ciò ha la seguente segnatura: *override fun popolateView(context: Context, view: RecyclerView)*

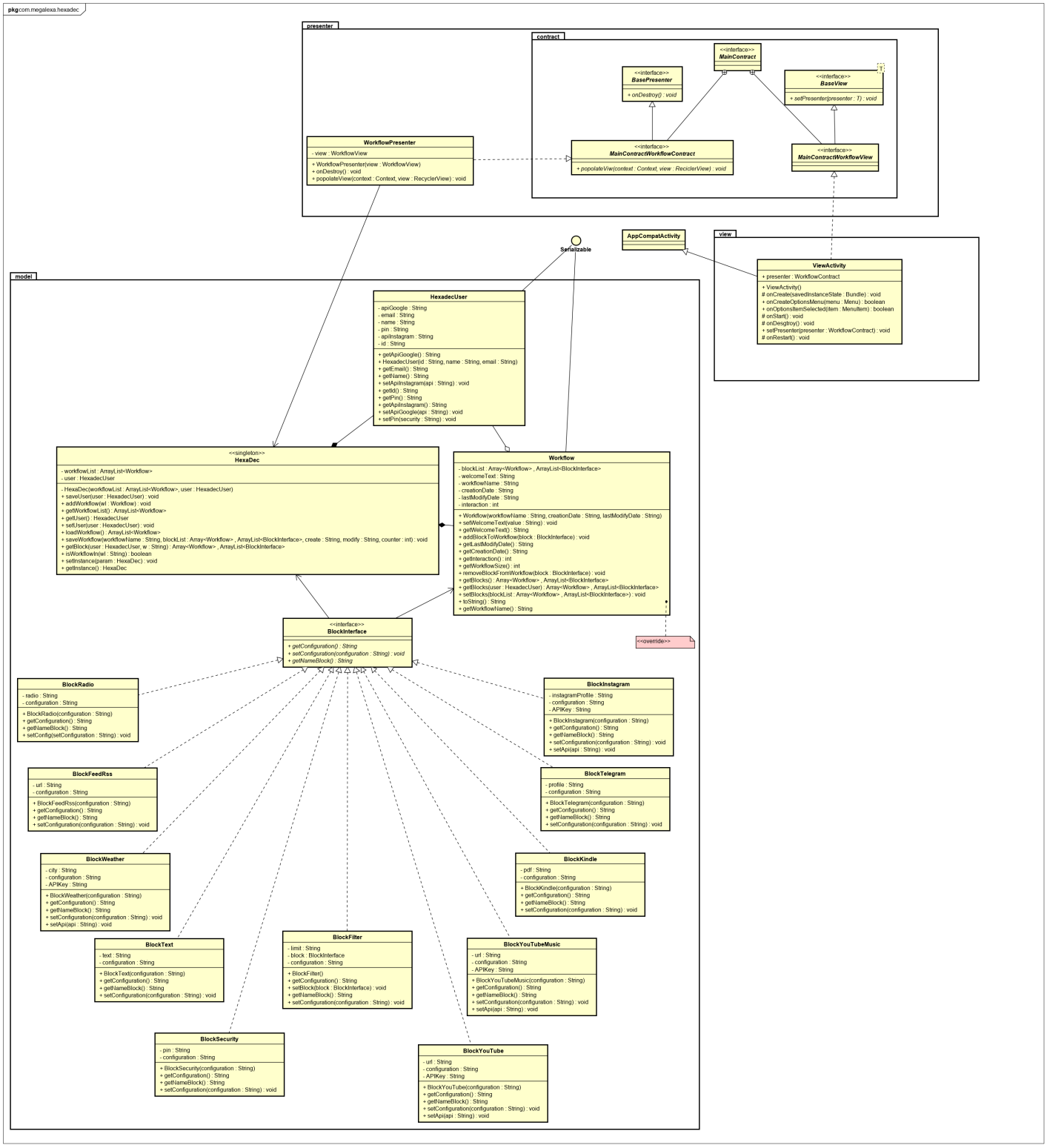


Figura 14: Diagramma delle classi: ViewActivity

5.6.3 SettingsActivity

E' la pagina di impostazione dell'applicazione HexadecApp. Questa vista visualizza alcune informazioni dell'utente quali la mail e il nome e permette di modificare il proprio PIN e di cambiare la lingua dell'applicazione. Il blocco sicurezza può essere aggiunto al workflow dall'activity AddBlockActivity

però la sua configurazione avviene solamente dalla pagina d'impostazioni, perchè il pin è univoco per tutti gli workflow che hanno al loro interno un blocco sicurezza.

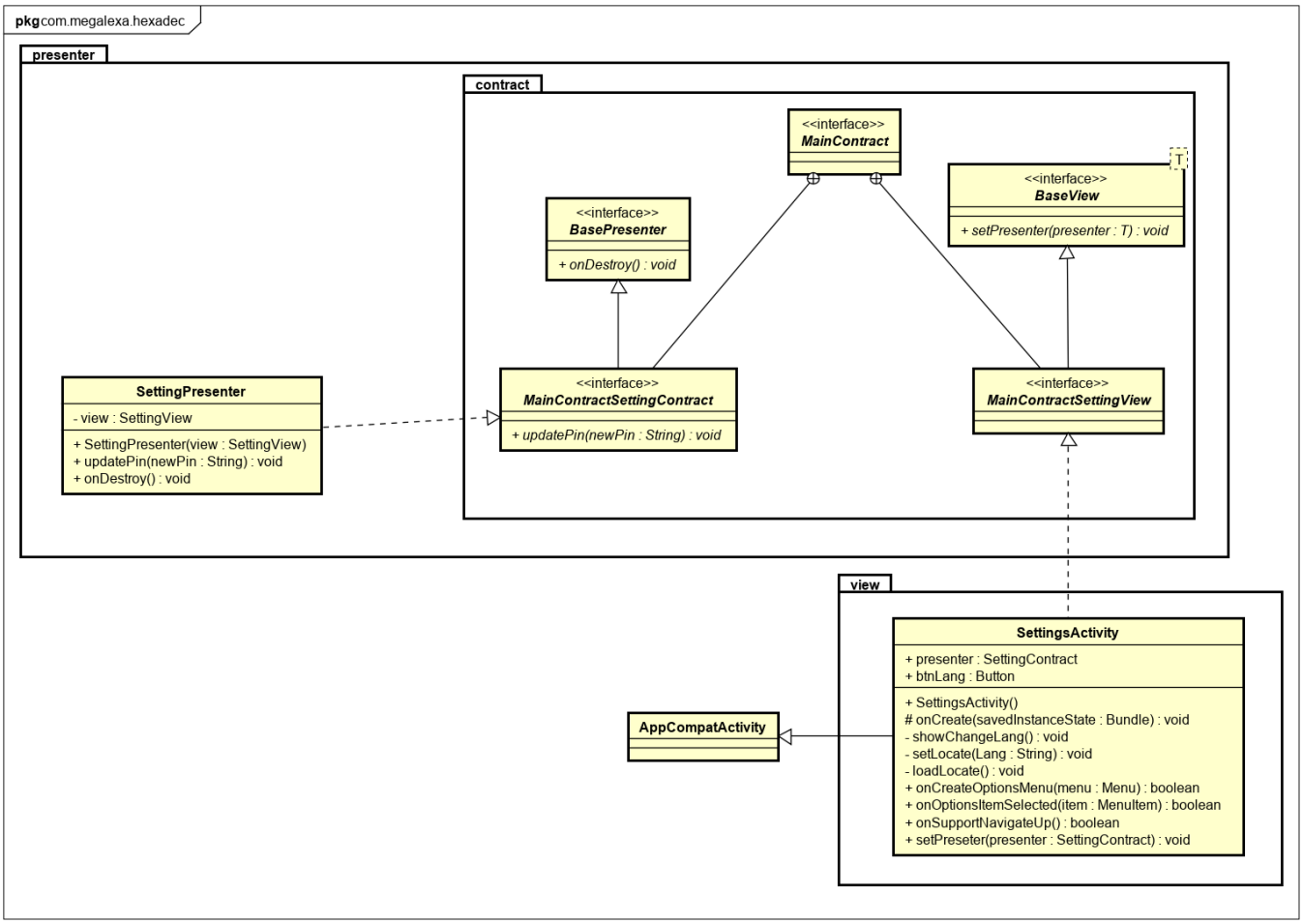


Figura 15: Diagramma delle classi: SettingsActivity

5.6.4 AddBlockActivity

In questa pagina è possibile selezionare il blocco da configurare ed inserirlo nel workflow. I blocchi configurati vengono aggiunti in un'oggetto Workflow passato tra le activity in modo da poter avere sempre l'oggetto aggiornato con i blocchi inseriti. Lo scambio di dati tra model e view viene gestito dal presenter *BlockPresenter* che definisce 2 metodi utili per il salvataggio dei blocchi nel workflow corrente e per il salvataggio del workflow a database. Di seguito viene analizzato il metodo *SaveWorkflow*:

- il metodo ha la seguente segnatura *override fun saveWorkflow(workflow: Workflow)*. Viene utilizzata la keyword override per far notare al compilatore che il metodo è una sovrascrittura del metodo definito in BlockPresenter. All'interno del metodo viene preparato un oggetto *Json* da poter inviare al metodo put con le seguenti informazioni del worfklow:
  - IDUser;
  - WorkflowName;
  - WelcomeText;
  - SuggestedTime.

Il metodo put scrivere prepara l'oggetto Json da inviare alle lambda che scriverà a database le informazioni necessarie per la creazione del workflow.

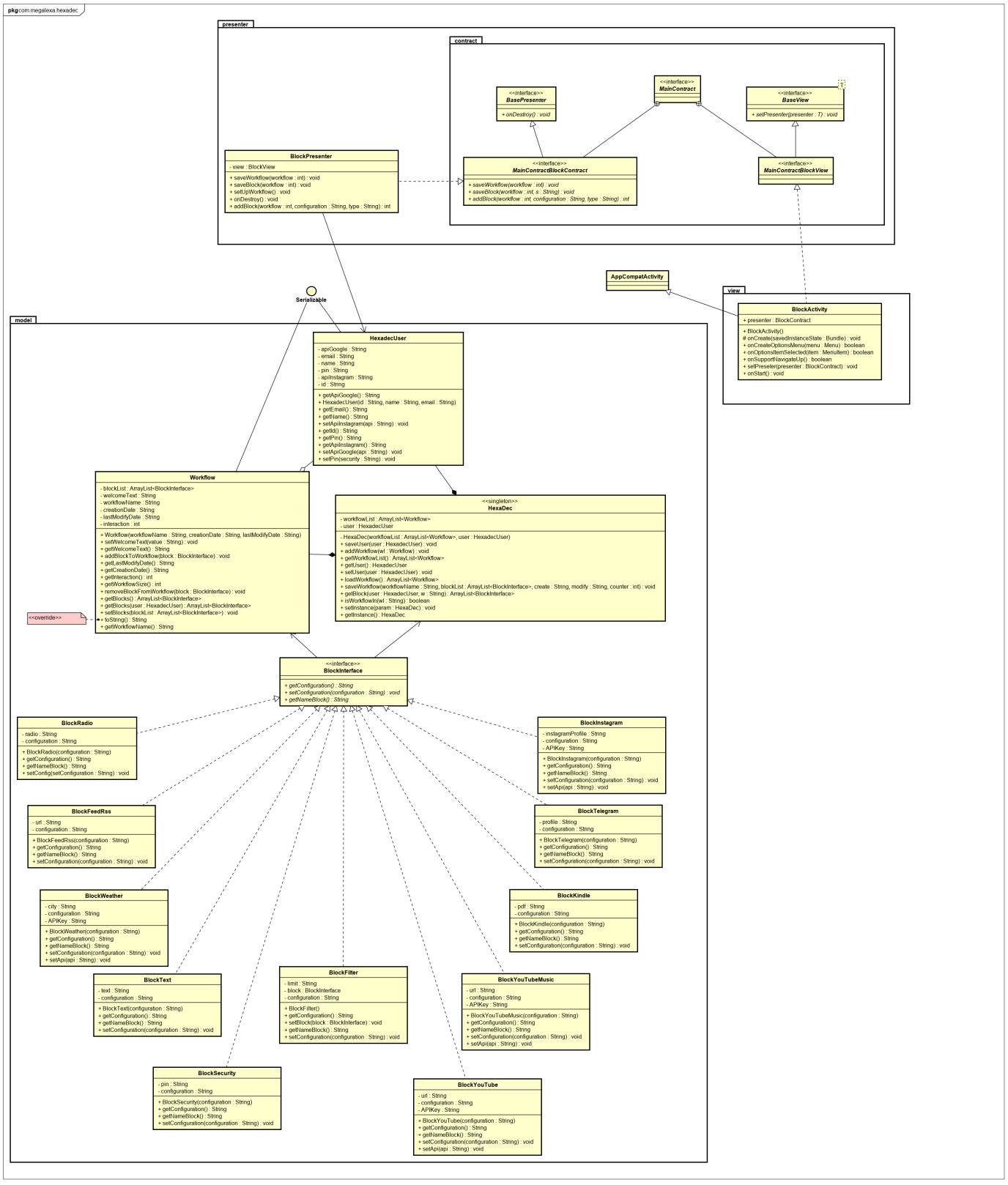


Figura 16: Diagramma delle classi: BlockActivity

5.7 Design Pattern

5.7.1 Model View Presenter

5.7.1.1 Scopo
Il pattern architetturale Model View Presenter (MVP) separa la vista dalla sua logica di presentazione per consentire a ciascuno di variare in modo indipendente.

5.7.1.2 Benefici
Abbiamo adottato questo pattern per i seguenti motivi:

- chiara separazione dell'interfaccia utente dalla logica dei dati;
- semplifica la progettazione dell'interfaccia utente;
- aumenta la manutenibilità dell'interfaccia utente perchè a causa dell'accoppiamento lasco tra View e Model.

5.7.1.3 Struttura
Abbiamo suddiviso l'applicativo mobile in 3 package:

- **model**: rappresenta la logica dei dati. Contiene al suo interno il package *blocks*, che raccoglie la definizione dei blocchi e delle relative risorse. All'interno del model viene definito il design pattern Singleton che verrà trattato nella prossima sezione;
- **view**: rappresenta l'interfaccia utente. Ogni view rappresenta una specifica activity che disegna sullo schermo gli elementi definiti nel layout xml associato, grazie ai quali l'utente può interagire con l'applicazione. All'interno di ogni activity sono presenti i listener utili per spostarsi da un'activity ad un'altra e un metodo *getIntents* per recuperare i dati dall'activity precedente.
- **presenter**: fa da mediatore tra la vista e il modello. Elabora i dati del Model in modo che possano essere visualizzati nella View e risponde alle richieste dell'utente nella View interagendo con il Model. Le view e i presenter sono legati da un contratto, spiegati dettagliatamente nella §5.3.

5.7.2 Singleton

5.7.2.1 Scopo
Il Singleton ha lo scopo di assicurarsi che una classe abbia solo un'istanza e fornire un punto di accesso globale a essa.

5.7.2.2 Benefici
Abbiamo adottato questo pattern per i seguenti motivi:

- singleton impedisce ad altri oggetti di istanziare la loro copia dell'oggetto Singleton, assicurando che tutti gli oggetti di accesso alla singola istanza.;
- **flessibilità**: dal momento che la classe controlla le istanze del processo, la classe ha la flessibilità necessaria per modificare il processo di creazione di un'istanza.

5.7.2.3 Struttura
Tutte le informazioni dell'utente autenticato vengono salvate all'interno dell'singleton, quindi abbiamo *Hexadec.User* per ricavare l'id dell'utente e *Hexadec.WorkflowList* per la sua lista worfklow. Il Singleton si avvale di un metodo *setInstance(param : HexaDec)* per settare l'istanza la prima volta e un metodo *fun getInstance()* che ritorna l'istanza corrente.

6 Architettura skill Amazon Alexa

6.1 Configurazione skill

La skill viene ospitata sul sito https://developer.amazon.com/it/alexa al quale è necessario autenticarsi per potervi accedere. Perché la skill funzioni è necessario collegarla con AWS Lambda, nello specifico con la funzione lambda che gestisce il *back-end*G, senza questo collegamento la skill non può funzionare.

Il seguente diagramma rappresenta l'architettura della skill:

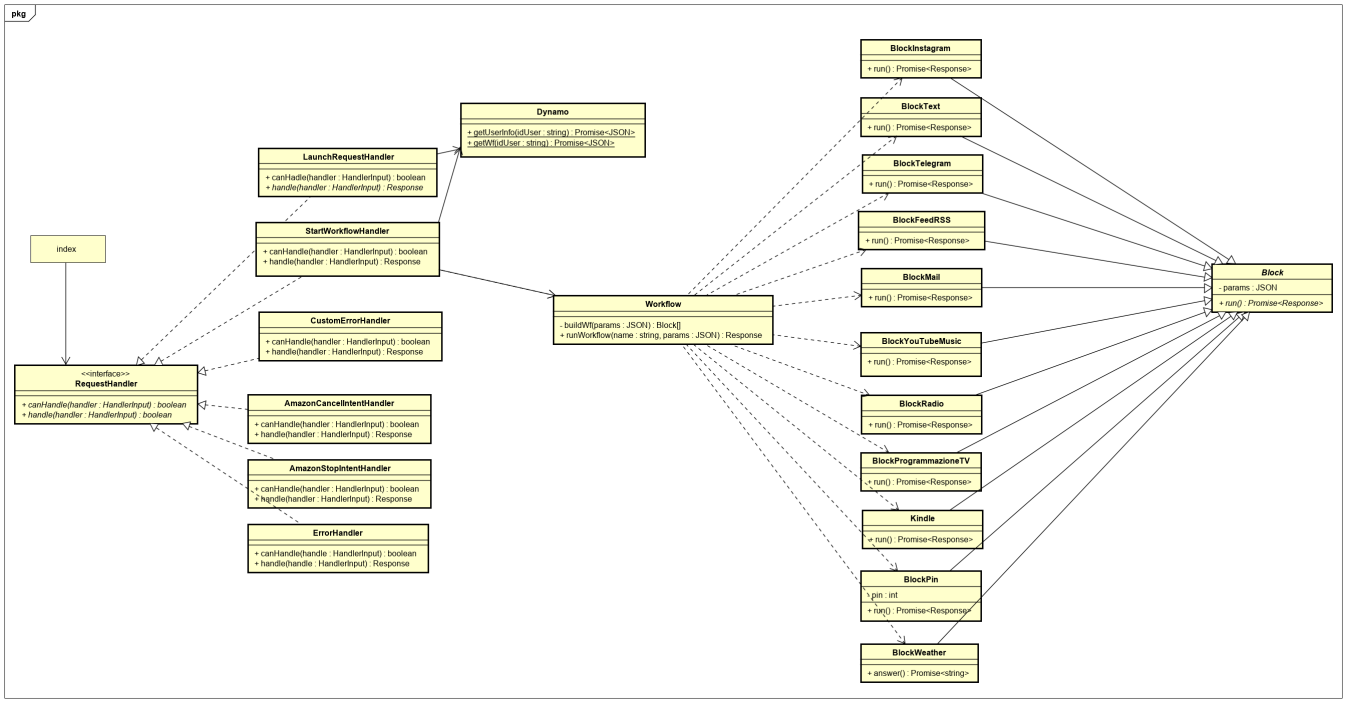


Figura 17: Architettura della skill

6.1.1 ARN e skillID

Per effettuare il collegamento seguire i seguenti passi:

- accedere a https://developer.amazon.com/it/alexa;
- accedere alla funzione lambda presente nella console AWS Lambda;
- copiare l'*ARN(Amazon Resource Name)*G sulla sezione "Endpoints" della skill alla voce "Default region";

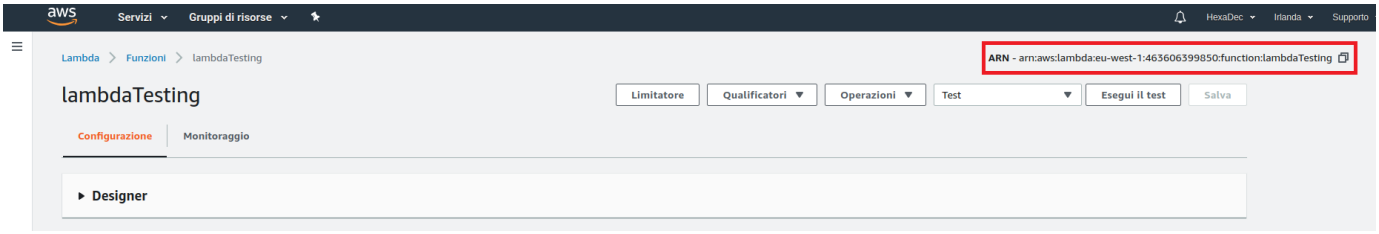


Figura 18: ARN Lambda

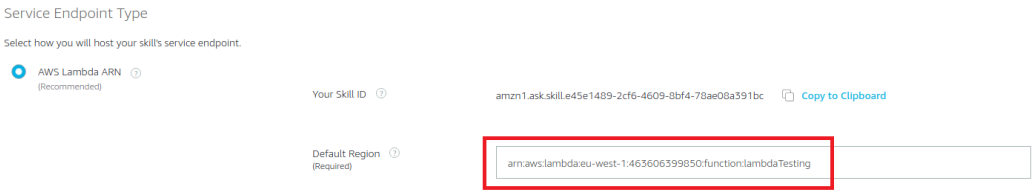


Figura 19: ARN Default region

- dalla sezione ”Endpoints” copiare lo skillID della skill e aggiungerlo come trigger per la funzione lambda di riferimento alla voce ”Alexa skill kit”.

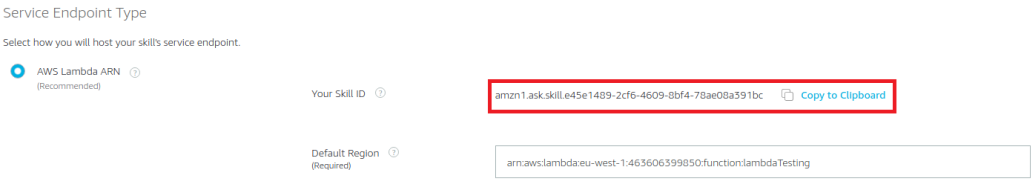


Figura 20: SkillID

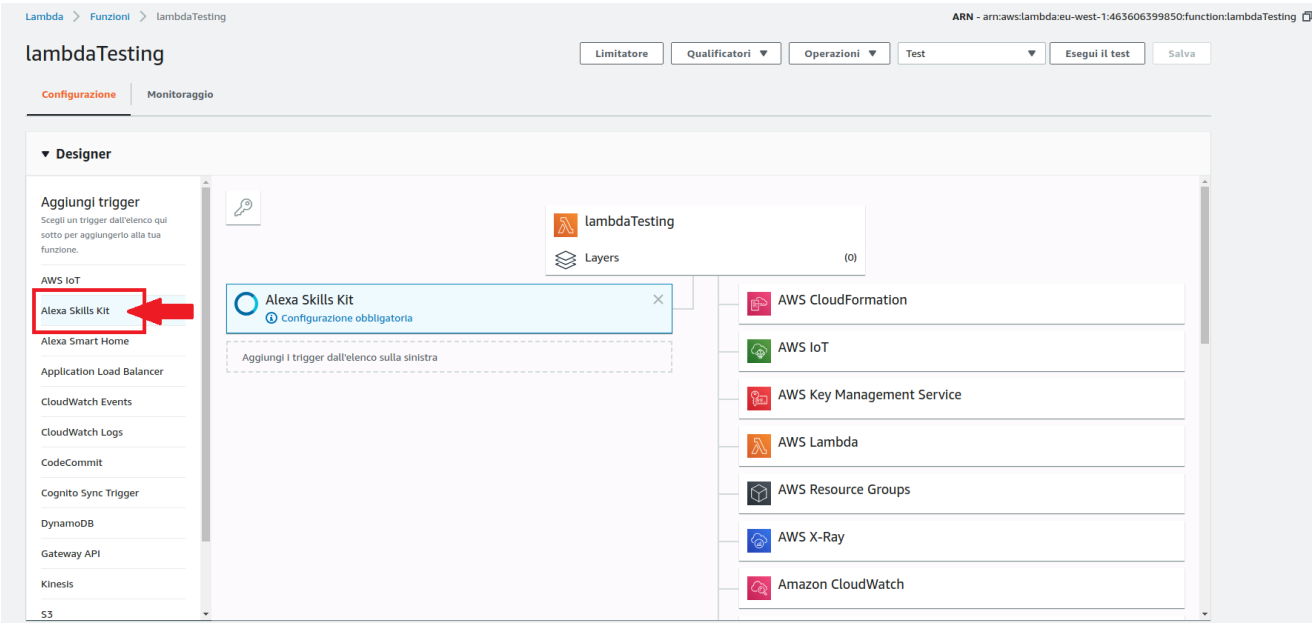


Figura 21: Selezione Alexa Skill Kit

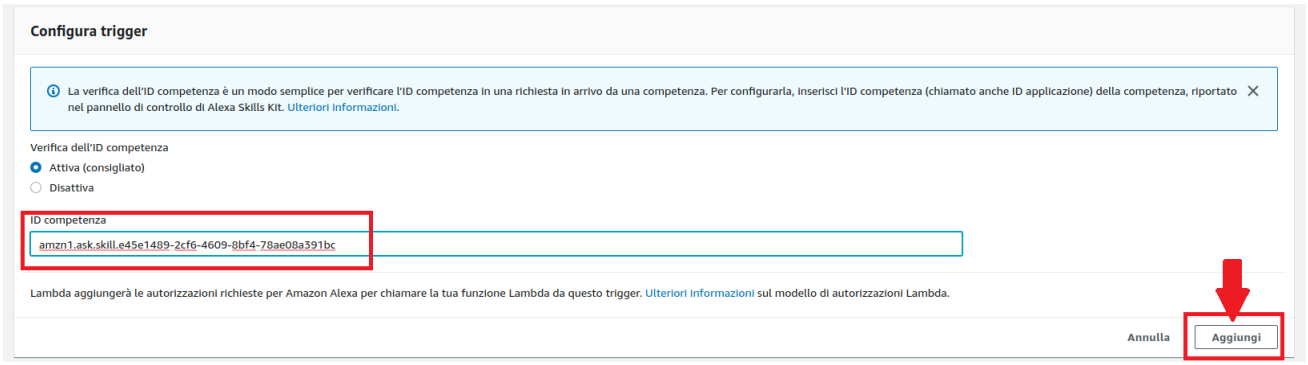


Figura 22: Inserimento SkillID

6.2 Back-end

6.2.1 Index

La logica della skill risiede nel file *index.ts* che si occupa di soddisfare le richieste dell'utente fatte ad Alexa interagendo con altri file .ts specifici per ogni sua domanda. Le modalità con cui opera l'index sono di ricevere dei dati contenuti in un file JSON generato in automatico alla richiesta dell'utente e in base a quali essi siano eseguire la funzione handler di competenza. I dati fondamentali che l'index deve ricevere per poter operare sono il tipo dell'*intent*G ed eventuali *slots*G.

6.2.2 Handler

RequestHandler è un'interfaccia che viene fornita dalle librerie di AWS incluse nel package 'aws-sdk'. Ogni classe che che implementa questa interfaccia viene definita classe handler che gestisce una specifica richiesta. Ogni handler contiene due metodi per la gestione dell'intent ricevuto:

- *canHandle(handlerInput)* si occupa di verificare se quello è l'handler di competenza per risolvere la richiesta dell'utente. La verifica avviene controllando il tipo di intent e talvolta il nome. In caso di match verrà ritornato un valore true e poi verrà eseguito il metodo *Handle*, in caso contrario ritornerà un false;
- *Handle(handlerInput)* si occupa del soddisfacimento della richiesta elaborando l'input e ritornando un output eseguibile da Alexa.

Gli handler da noi utilizzati sono:

- **LaunchRequestHandler** è l'invocazione iniziale della skill; viene attivato da una frase di apertura come 'Alexa, open MegAlexa' e dopo aver controllato se l'utente è autenticato ritorna un messaggio di benvenuto in caso affermativo, altrimenti, invita l'utente ad autenticarsi. Handler attivato da un intent di tipo LaunchRequest;
- **StartWorkflowHandler** viene attivato quando l'utente richiede l'esecuzione di un determinato workflow. Prende i dati del workflow richiesto tramite una chiamata al database. Handler attivato da una IntentRequest dal nome startWorkflowIntent;
- **ConfirmedWorkflowHandler** viene attivato quando l'utente conferma l'esecuzione del workflow proposto dall'intelligenza artificiale. Il nome dell'intent è ConfirmedWorkflow;
- **DenyIntentHandler** viene attivato quando l'utente rifiuta l'esecuzione del workflow proposto dall'intelligenza artificiale. Il nome dell'intent è DenyIntent;

- **PinWorkflowHandler** viene attivato quando l'utente dice il pin richiesto dalla skill per l'esecuzione di un workflow. Il nome dell'intent è PinWorkflowIntent;
- **StopIntentHandler** è un handler di default che elabora la richiesta di arresto della skill da parte dell'utente. Handler attivato da intent quali StopIntent e HeplIntent;
- **SessionEndedHandler** viene inviato quando la sessione attualmente aperta della skill è chiusa perché l'utente è uscito dalla skill, l'utente ha fornito una risposta che la skill non può comprendere o si verifica un errore;
- **CustomErrorHandler** è l'handler dedicato alla gestione di errori. Si attiva quando nessuno degli altri handler è stato attivato.

Gli handler sono strutturati con uno *strategy pattern*G descritto nel seguente diagramma:

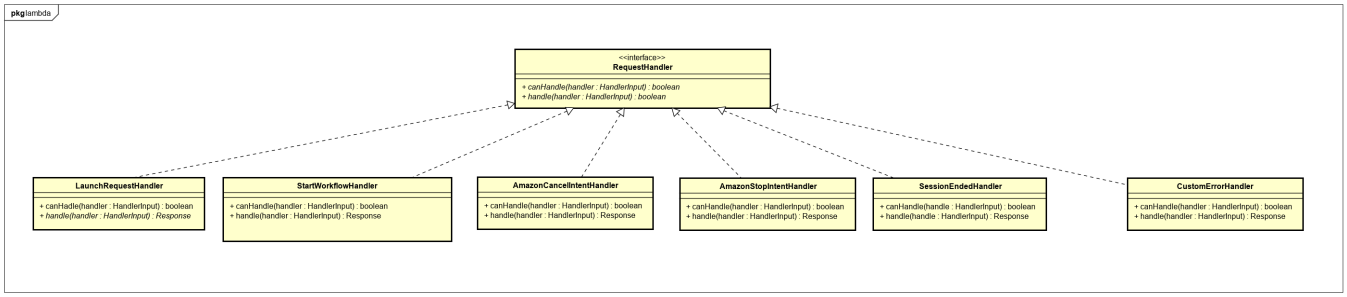


Figura 23: Diagramma delle classi: Handlers

HandlerInput è il file JSON automaticamente creato al momento della richiesta che contiene tutte le informazioni necessarie. Un esempio di utilizzo dell'HandlerInput è handlerInput.requestEnvelope.request.type per capire il tipo dell'intent. Nel caso in cui nessun handler soddisfi la richiesta dell'utente, viene invocato il EndSessionRequestHandler, che è un handler di default, che tornerà un messaggio di errore ad Alexa.

6.2.3 Block

Block è una classe astratta che definisce la struttura dei blocchi dei workflow. Espone un metodo *run()* che verrà chiamato per l'effettiva esecuzione del blocco specifico. La sua ridefinizione delinea il comportamento del blocco concreto. I blocchi che implementano Block sono i seguenti:

- BlockText: consente ad Alexa di leggere un testo;
- BlockFeedRSS: consente di ricevere e leggere un feedRSS impostato da un utente;
- BlockMail: consente di leggere le ultime mail non lette da Gmail;
- BlockRadio: consente di riprodurre un canale radio selezionato;
- BlockProgrammazioneTV: consente di riferire la programmazione serale del canale TV selezionato;
- BlockKindle: consente di leggere un file .pdf;
- BlockPin: consente l'avvio o meno di un workflow in base alla corretta dicitura di un codice di sicurezza;
- BlockWeather: consente di sapere le condizioni meteo di una località selezionata o tramite geolocalizzazione;

- BlockInstagram: consente di collegarsi tramite l'account Instagram e vedere le proprie ultime foto.

6.2.4 Workflow

La classe Workflow costruisce oggetti concreti di Block tramite l' *abstract factory*G grazie a un parametro di tipo JSON params passato alla costruzione dell'oggetto di tipo Workflow. Su questo oggetto viene svolto uno switch case che ritorna l'azione del blocco corretto.

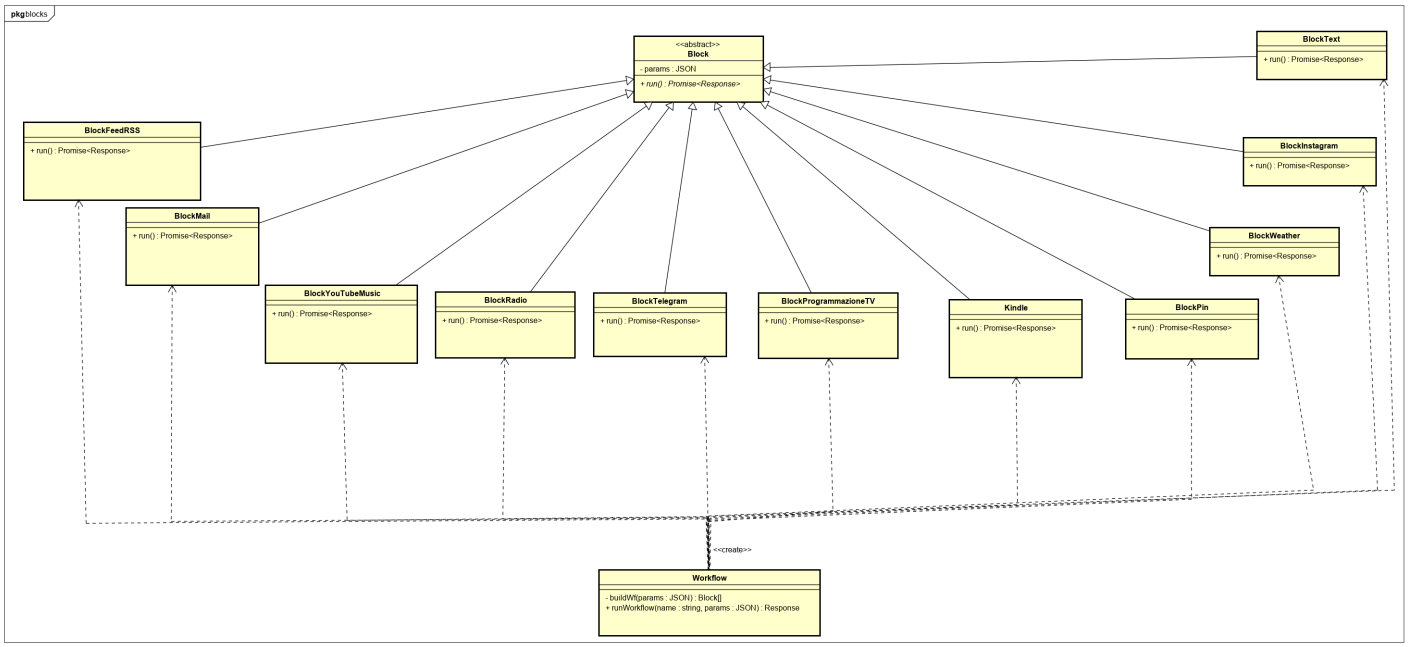


Figura 24: Diagramma delle classi: Blocks

6.2.5 Dynamo

La connessione al database è gestita dalla classe Dynamo tramite il pattern *Singleton*G. I servizi AWS si occupano di gestire le multiple richieste di accesso a DynamoDB. Essa fornisce due funzionalità:

- *getWorkflow (work : string)* : restituisce il workflow definito dal parametro work di un determinato utente;
- *getAllWorkflow ()* : fornisce tutti i workflow dell'utente collegato alla skill in formato JSON;
- *getAllWorkflowOrderBy(param :string)* : restituisce tutti i workflow di un utente ordinati secondo il parametro passato al metodo;
- *getUserInfo()* : fornisce tutte le informazioni memorizzate su database dell'utente che fatto una richiesta alla skill in formato JSON.

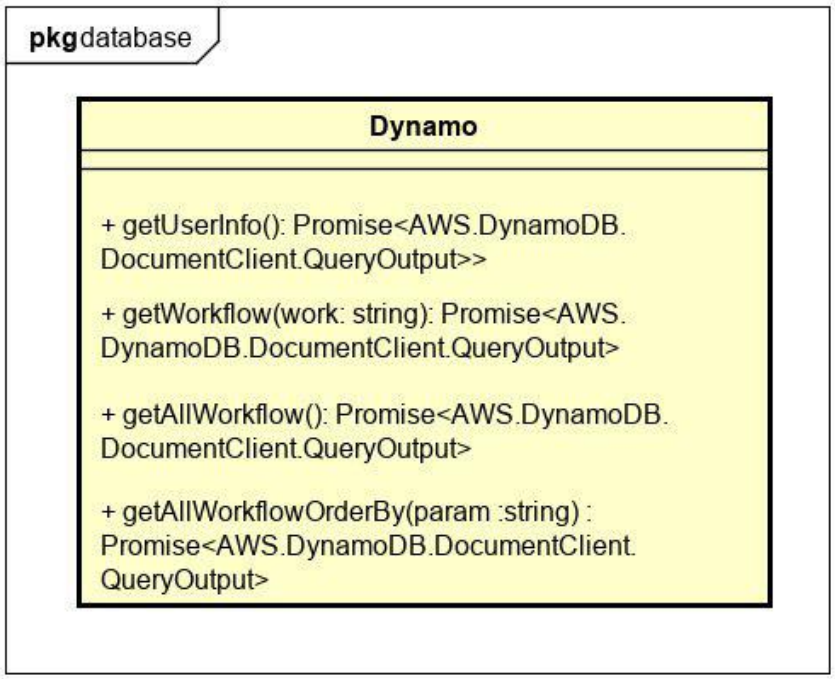


Figura 25: Diagramma della classe: Dynamo

6.3 Design Patterns

I design pattern utilizzati per l'architettura della skill sono il singleton per la connessione a database, l'abstract factory per la gestione di workflow e blocchi e lo strategy pattern per gli handler.

6.3.1 Singleton

Questo design pattern viene applicato per la gestione della connessione al database. Vi è un'unica istanza della classe 'DB' che viene costruita partendo dall'informazione dell'id user dell'utente che sta utilizzando la skill. Questa istanza permette di accedere ai dati dell'utente contenuti nel database attraverso delle specifiche funzioni lambda. L'utilizzo di questo design pattern è stato scelto per la possibilità di accedere da qualsiasi parte alla singola istanza del database. Poichè le chiamate al database sono molteplici e in una singola sessione si riferiscono sempre allo stesso utente e quindi agli stessi dati, il singleton è il design pattern che meglio si addice per la funzione del database.

6.3.2 Abstract factory

Questo design pattern ha lo scopo di fornire un'interfaccia 'Block' utile per creare famiglie di prodotti quali sono i blocchi poi implementati. L'interfaccia presenta un metodo run che viene implementato da ogni blocco. L'abstract factory è il dsign pattern che meglio permette di rappresentare la struttura dei workflow. Un'interfaccia generica dalla quale si concretizzano le classi figlie che costituiscono le funzionalitò che la skill offre.

6.3.3 Strategy pattern

Gli handlers sono delle classi che differiscono solo per il loro comportamento. La necessità di varianti dello stesso algoritmo per cui i client non devono preoccuparsi rispecchia le caratteristiche dello strategy pattern che si rivela la scelta migliore per gli handlers. Infatti con il suo utilizzo viene definita una famiglia di algoritmi interscambiabili e inidpendenti dal client.

7 Estensione delle funzionalità

7.1 App

L'estensione dell'applicazione avviene mediante l'aggiunta di nuovi blocchi. Per una corretta estensione seguire i seguenti passi:

- Definire un blocco che derivi dall'interfaccia *BlockInterface*;
- Il nuovo blocco deve implementare i metodi dell'interfaccia *BlockInterface*:
  - *getConfig():* ritorna la configurazione attuale del blocco;
  - *getNameBlock():* ritorna il nome del blocco;
  - *setConfig(configuration:String):* setta la configurazione del blocco.
- Recarsi nella cartella **presenter / connection / block** e creare il nuovo file connection che estende l'interfaccia *blockConnection* e sovrascrivere i seguenti metodi sostituendo newBlockName con il nome effettivo del blocco:
  - *fun convertFromJSON(jsonObject: JSONObject):newBlockName*;
  - *fun <NewBlockName>convertToJSON(t: newBlockName): JSONObject*.
  - *val resource: String get() = "blocks/newBlockName"*.
- Recarsi nella console di *AWS API Gateway* e creare le nuove risorse per il nuovo blocco avente come risorsa di primo livello *block/newBlock*.
- Definire un widget Activity che rappresenti l'inserimento dei dati da parte dell'utente;
- Inserire il listener dell'elemento aggiunto alla lista nel widget appena creato;
- Recarsi nella cartella textbfpresenter / adapter e creare una nuova classe dentro il file recyclerViewConfigurationAdapter.kt. La classe deve avere la seguente segnatura:
  - *class newBlockName(val nomeT: String, val positionTW: String,* ***listaParametri)*** *: RowType*
    Nota Bene: Sostituire listaParametri con il nome dei campi dati che vuoi visualizzare nella RecyclerView.
- Aggiornare il AddBlockPresenter e il ConfigurationBlockPresenter in modo che supporti l'aggiunta del blocco appena creato.

7.2 Front-end

Per modificare il layout e i colori delle viste sarà necessario agire sui seguenti file XML:

- **res/layout** : in questa cartella sono presenti i file xml che definiscono il layout delle view e delle RecyclerView. Per aggiungere o togliere widget è possibile seguire la guida ufficiale di Android disponibile al seguente link: https://developer.android.com/guide/topics/ui
- **res/value/strings** : in questa cartella sono presenti i due file strings.xml e strings(it).xml che definiscono tutte le stringhe tradotte per ciascun lingua dell"applicazione. Nel caso si volesse cambiare la traduzione di alcune stringhe oppure si volesse semplicemente aggiungere o rimuovere delle stringhe lo si può fare agendo su questi file.

  Nota Bene: Quando si effettua la traduzione di una stringa, l'id di quest'ultima dev'essere presente in entrambi i file come nel seguente esempio:

  - strings(it).xml: *<string name="newWorkflowtButton" >Salva workflow</string>*
  - strings.xml: *<string name="newWorkflowtButton" >Save workflow</string>*

- **res/value/colors:**: questo file definisce tutti i colori delle varie componenti dell'applicazione.
- **res/value/styles:**: questo file definisce una collezione di attributi che specificano lo stile della view. Tra gli attributi modificabili ci sono la grandezza e il colore del font oppure il colore di background.

7.3 Skill

La skill può essere estesa nei seguenti modi:

- Aggiunta di nuovi blocchi;
- Aggiunta di nuove frasi per Alexa.

7.3.1 Aggiunta di blocchi

Per una corretta estensione seguire i seguenti passi:

- Definire una classe che rappresenta il blocco che derivi dalla classe astratta*Block* e implementando i metodi rispettando le loro firme;
- Nella classe *Workflow* aggiungere il case che gestisca il nuovo blocco creato.

7.3.2 Aggiunta di nuove frasi per Alexa

Per una corretta estensione seguire i seguenti passi:

- Andare sul seguente link : https://developer.amazon.com/alexa/console/ask;
- Selezionare la skill;
- Andare sull'intent StartWorkflowIntent;
- Aggiungere le nuove frasi.

8 Test

8.1 Test in ambiente AWS

Per testare la è necessario accedere al portale sviluppatori di Alexa e selezionare la skill da 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 è disponibile un Alexa fisico, possono essere eseguiti i test direttamente su di esso.

9 Licenza

A Glossario interno

A

API Gateway
Servizio offerto da Amazon che permette la creazione, manutenzione e protezione di API su qualsiasi scala.

ARN
Amazon Resource Name identifica in modo univoco una risorsa AWS.

D

DBMS
Le principali funzioni dei DBMS sono quelle di garantire il mantenimento della corretta strutturazione dei dati nei diversi database gestiti e di facilitare l'accesso delle applicazioni ai dati, tramite opportune istruzioni impartite al sistema operativo.
DynamoDB
Servizio di database NoSQL offerto da Amazon che supporta i modelli di dati di tipo documento e di tipo chiave-valore.

F

Factory pattern
Definisce un'interfaccia per la creazione di un oggetto, lasciando però alle sottoclassi la decisione riguardo a quale classe istanziare. Tramite il Factory Method si fa in modo che una classe possa delegare l'istanziazione di un prodotto alle sue sottoclassi.

Framework
È una libreria di codice o una porzione di codice prestabilita che lo sviluppatore può utilizzare per implementare alcune specifiche funzioni.

G

General purpose
Sono quei linguaggi che possono essere usati per affrontare qualunque tipo di problema/applicazione.

I

Intent
Un intent rappresenta un'azione attivata dall'utente tramite un comando vocale.

J

JetBrains
È un'azienda di sviluppo software ceca nata nel 2000. L'azienda offre una vasta gamma di ambienti di sviluppo integrati (IDE) per i linguaggi di programmazione Java, Ruby, Python, PHP.

K

Kotlin
È un linguaggio di programmazione orientato agli oggetti usato per sviluppare applicazioni Android.

L

Lambda
Servizio di elaborazione serverless che esegue il tuo codice in risposta a determinati eventi e gestisce automaticamente le risorse di elaborazione sottostante.

Linguaggio di programmazione libero
E' un linguaggio formale generato da una grammatica non contestuale, ovvero tale che le cui regole agiscono su simboli non terminali a prescindere dal contesto in cui essi appaiono.

M

Multi-paradigma
Vul dire che un certo linguaggio di programmazione supporta più tipi di programmazione (programmazione ad oggetti, funzionale)

O

Open source
Un software di cui i detentori dei diritti rendono pubblico il codice sorgente, favorendone il libero studio e permettendo a programmatori indipendenti di apportarvi modifiche ed estensioni.

S

Skill
Una Skill è una capacità o abilità di Alexa. Alexa mette a disposizione delle skill di default (come ad esempio la possibilità di riprodurre musica), inoltre i programmatori possono aggiungerne delle altre configurandone di nuove.

Slots
Per ogni intento, puoi specificare i parametri che indicano le informazioni necessarie all'intento per adempiere alla richiesta dell'utente. Questi parametri sono gli slots.

Strategy pattern
Il pattern Strategy è utilizzato quando occorre definire una famiglia di algoritmi e renderli inter- cambiabili, permettendo loro di variare indipendentemente dal client. Oggetti Strategy possono essere utilizzati per fornire diverse implementazioni di uno stesso algoritmo o comportamento, e il client può scegliere il più appropriato in base ai diversi trade-off di tempo e spazio.

W

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

World Wide Web Consortium

È un'organizzazione non governativa internazionale che ha come scopo quello di sviluppare tutte le potenzialità del World Wide Web.