

Università degli Studi di Padova
DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"
CORSO DI LAUREA IN INFORMATICA



**Concierge Croccante: accoglienza dell'ospite
mediante assistente Amazon Alexa**

Tesi di laurea triennale

Relatore

Prof. Paolo Baldan

Laureando

Matteo Pellanda

ANNO ACCADEMICO 2018-2019

Matteo Pellanda: *Concierge Croccante: accoglienza dell'ospite mediante assistente Amazon Alexa*, Tesi di laurea triennale, © Settembre 2019.

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage dal laureando Matteo Pellanda presso l'azienda Crispy Bacon Srl. Lo stage è stato svolto al termine del percorso di studi della laurea triennale in Informatica, ed ha avuto la durata di trecentoventi ore.

L'obiettivo è stato quello di realizzare una Skill per l'assistente vocale Amazon Alexa, utilizzando il linguaggio NodeJS. Prima della sua realizzazione è stato redatto un documento di analisi del prodotto e alla fine una documentazione sulle tecnologie software utilizzate.

Il presente documento ha lo scopo di illustrare il contesto aziendale dove è stato svolto lo stage, le attività svolte, ed infine una valutazione sul lavoro effettuato. La Skill realizzata ha nome Concierge Croccante, che nel documento potrà essere abbreviato con l'acronimo C.C.

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Paolo Baldan, relatore della mia tesi, per l'aiuto e il sostegno fornитоми durante la stesura.

Desidero ringraziare con affetto i miei genitori per il sostegno, il grande aiuto ricevuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Inoltre desiderio ringraziare i miei amici e compagni di corso per tutti i bellissimi anni passati insieme e le mille avventure vissute.

Un grazie anche a Valentina, la persona che mi è stata particolarmente vicina in quest'ultimo anno accademico.

Tezze sul Brenta, Settembre 2019

Matteo Pellanda

Indice

1 Il contesto aziendale	1
1.1 Profilo aziendale	1
1.2 Dominio applicativo	2
1.2.1 Alexa	2
1.2.2 Skill	2
1.2.3 Amazon Web Service	2
1.2.4 L'idea	3
1.2.5 Interesse aziendale	3
1.2.6 Il progetto	3
1.3 Metodologia aziendale	3
1.4 Tecnologie utilizzate	5
1.4.1 Node.js	5
1.4.2 Alexa Presentation Language	5
1.4.3 Servizi Amazon Web Service	5
1.4.4 Google Calendar	5
1.4.5 Slack	6
1.5 Strumenti utilizzati	6
1.5.1 Alexa Developer Console	6
1.5.2 Amazon Echo Show	6
1.5.3 NPM	7
1.5.4 Gitlab	7
1.5.5 Ambiente di sviluppo	7
1.6 Organizzazione del testo	7
2 Analisi dei Requisiti	9
2.1 Obbiettivo	9
2.1.1 Cliente finale	9
2.2 Utenti coinvolti	10
2.3 Requisiti richiesti	10
2.3.1 Identificazione dei Requisiti	10
2.4 Casi d'uso	13
2.4.1 Attori	13
2.4.2 Casi d'uso - Visitatore	14
2.4.3 Casi d'uso - Postino/Corriere	19
2.5 Voice User Interface - VUI	22
2.5.1 Voice Flow	23
2.5.2 Interazione vocale - UC6	25
2.6 Graphical User Interface - GUI	26

2.7	Limiti	28
3	Progettazione e codifica	29
3.1	Architettura	29
3.2	Configurazione servizi Amazon Web Service	29
3.2.1	AWS SES	29
3.2.2	AWS S3	31
3.2.3	AWS IAM	32
3.2.4	AWS CloudWatch	34
3.2.5	AWS DynamoDB	34
3.3	Creazione della Skill	36
3.3.1	Alexa Developer Console	36
3.3.1.1	Skill ID	38
3.3.2	AWS Lambda	38
3.3.2.1	Endpoint	40
3.4	Organizzazione del codice	41
3.4.1	Handlers	42
3.4.2	Utils	44
3.5	Pacchetti	46
3.6	Calendario	48
3.6.1	Google Calendar	48
3.6.1.1	OAuth2	48
3.6.1.2	Lettura eventi	51
3.7	Invio notifiche	54
3.7.1	Slack	54
3.7.1.1	Incoming Webhooks	54
3.7.1.2	Invio notifiche	55
3.7.2	E-mail	60
3.7.2.1	Invio e-mail	60
3.8	Lettura del Database	62
3.9	Async e Await	63
4	Realizzazione e Testing	65
5	Conclusioni	67
5.1	Risultato ottenuto	67
5.2	Analisi critica del prodotto	67
5.3	Analisi critica del lavoro	67
5.4	Valutazione degli strumenti utilizzati	67
5.5	Possibili miglioramenti	67
5.6	Possibili estensioni	67
A	Appendice A	69
	Bibliografia	73

Elenco delle figure

1.1	Logo Crispy Bacon srl	1
1.2	Logo di Alexa e AWS	3
1.3	Modello Agile - SCRUM	4
1.4	Architettura progetto C.C.	6
2.1	Utenti del sistema	13
2.2	Casi d'uso visitatore aente appuntamento	14
2.3	Sotto casi d'uso visitatore aente appuntamento	17
2.4	Sotto casi d'uso visitatore aente appuntamento	18
2.5	Casi d'uso corriere	19
2.6	Esempio di conversazione	22
2.7	Diagramma di attività VUI - Concierge Croccante	24
2.8	Esempio GUI - Home	26
2.9	Esempio di visual - Aiuto	27
2.10	Esempio di visual - Consegna	27
2.11	Esempio di visual - Visitatore aente appuntamento	27
3.1	Icona AWS SES	30
3.2	Icona AWS S3	31
3.3	Icona AWS IAM	32
3.4	Esempio di Role creata	33
3.5	Icona AWS CloudWatch	34
3.6	Icona AWS DynamoDB	35
3.7	Esempio creazione tabella AWS DynamoDB	35
3.8	Esempio Skill ID	38
3.9	Selezione regione AWS Lambda	39
3.10	Esempio URL ARN Lambda	40
3.11	Esempio URL ARN Lambda	40
3.12	Codice funzione	40
3.13	Gerarchia cartella progetto C.C.	42
3.14	Gerarchia cartella progetto C.C. - Handlers	44
3.15	Gerarchia cartella progetto C.C. - Utils	46
3.16	Logo NPM	46
3.17	Icona Google Calendar	48
3.18	Icona servizio - Slack	54
3.19	Icona servizio - SES	60

Elenco delle tabelle

2.1	Tabella tracciamento requisiti obbligatori - VUI	11
2.2	Tabella tracciamento requisiti obbligatori - GUI	12
2.3	Tabella tracciamento requisiti obbligatori - Funzionalità	12
2.4	Tabella tracciamento requisiti desiderabili	12
2.5	Tabella casi d'uso visitatore avente appuntamento e non	16
2.6	Tabella sotto casi d'uso visitatore avente appuntamento e non	17
2.7	Tabella sotto casi d'uso visitatore avente appuntamento e non	18
2.8	Tabella casi d'uso corriere	21
2.9	Tabella tipologia dati per tipo di utente	23

Capitolo 1

Il contesto aziendale

1.1 Profilo aziendale

Crispy Bacon srl¹ è una software development company nata nel 2013 da quattro soci fondatori e che conta oggi più di 50 persone tra le sedi di Marostica (VI) e Milano. Una realtà "technology driven" (dipendere dalla tecnologia e utilizzarla in modo pratico), volta a cogliere le migliori tecnologie presenti sul mercato per offrire ai clienti strumenti digitali sempre più avanzati e innovativi, ne sono un recente esempio le applicazioni per smart speakers quali Amazon Alexa² e Google Home.

L'obiettivo di Crispy Bacon è quello di entrare nel mercato italiano con tecnologie di frontiera quali soluzioni bancarie, per l'industria e la moda. I principali servizi offerti rientrano nel campo dello sviluppo web e mobile, consulenza nell'ambito della digital transformation, UX/UI design e cloud computing, sempre al fianco dei propri clienti. Quello che Crispy Bacon si prefigge è di realizzare esperienze soddisfacenti mediante l'utilizzo di metodologie, approcci e tecnologie dirompenti.



Figura 1.1: Logo Crispy Bacon srl

¹Crispy Bacon srl. URL: <https://crispybacon.it>

²Amazon Alexa. URL: <https://developer.amazon.com/it/alexa>

1.2 Dominio applicativo

1.2.1 Alexa

Alexa³ è un assistente vocale intelligente basato su computabilità cloud sviluppato dalla sezione Lab126⁴ di Amazon utilizzato sui dispositivi in commercio quali Amazon Echo ed Echo Dot. Con Alexa, è possibile sviluppare performance vocali naturali che offrono ai clienti un modo più intuitivo per interagire con la tecnologia che utilizzano tutti i giorni. È in grado quindi di interpretare il linguaggio naturale e dialogare fornendo informazioni di diverso tipo. Le funzioni più comuni sono: riprodurre musica, gestire liste (della spesa o delle cose da fare), impostare promemoria e sveglie, effettuare streaming di brani musicali e podcast, riprodurre audiolibri e fornire previsioni meteorologiche, informazioni sul traffico e riprodurre altre informazioni in tempo reale, come le notizie. Alexa è in grado anche di controllare diversi dispositivi intelligenti, usando se stesso come sistema di automazione domestica per la gestione della domotica.

1.2.2 Skill

Come detto in precedenza, alcune funzioni di Alexa sono native, quali ricerche sul web, sveglie, liste, meteo, ecc.. Alexa fornisce inoltre delle Skill, ovvero dell'applicazioni di terze parti appositamente sviluppate in base alle necessità, che consentono di creare un'esperienza d'uso più personalizzata. Tali Skill possono essere installate nel proprio dispositivo su cui risiede l'assistente vocale ed avviate utilizzando il comando di lancio. Amazon mette quindi a disposizione molteplici servizi dedicati, tra cui la vendita all'interno dello store, per creare Skill in modo da rendere l'esperienza di utilizzo personale unica.

1.2.3 Amazon Web Service

Amazon Web Services Inc, nota ed abbreviata con la sigla AWS, è un'azienda di proprietà di Amazon che fornisce più di 165 servizi completi di cloud computing⁵ tra cui data center, email service e molto altro. Questi servizi offerti sono operativi in 20 paesi sparsi in tutto il mondo, a breve anche in Italia, che conta di arrivare a 24 aree geografiche entro il 2020. AWS offre molti servizi utilizzati per la realizzazione di questo progetto, tra i quali Amazon Lambda, DynamoDB e Amazon Simple Storage Service (S3), fornendo quindi soluzioni on-demand con caratteristiche di high availability⁶, ridondanza e sicurezza. Tali servizi possiedono combinazioni tra costo finale, caratteristiche, tempo di utilizzo e performances ottimali per utenti e le aziende. Tali caratteristiche rendono l'ecosistema di AWS grande e dinamico, con milioni di clienti attivi, di ogni settore e dimensione, incluse start-up, aziende e organizzazioni del settore pubblico in tutto il mondo.

³https://it.wikipedia.org/wiki/Amazon_Alexa

⁴Società americana di ricerca e sviluppo hardware per computer.

⁵Servizi on-demand da un fornitore ad un cliente finale attraverso la rete Internet

⁶Caratteristica di un sistema, che mira a garantire un livello concordato di prestazioni operative per un periodo superiore al normale.



Figura 1.2: Logo di Alexa e AWS

1.2.4 L'idea

Nasce da qui l'idea di voler realizzare, sfruttando l'ecosistema di AWS e l'assistente vocale Alexa, un concierge virtuale che possa, in maniera facile e pratica, accogliere clienti e visitatori senza l'intervento umano in maniera totalmente automatica.

1.2.5 Interesse aziendale

L'azienda desidera la sua realizzazione per uso personale all'interno degli uffici e come prototipo per poter sviluppare prodotti simili al fine di proporli ai clienti in maniera personalizzata o preconfezionata.

1.2.6 Il progetto

Nasce da questa idea il progetto Concierge Croccante, una Skill da poter installare sugli speakers Amazon con integrato l'assistente vocale Alexa. Essa permetterà di annunciarsi ed essere accolti al momento dell'ingresso presso un ufficio, innescando un processo di accettazione e di notifica agli interessati dell'arrivo dell'ospite.

1.3 Metodologia aziendale

Cripsi Bacon è da considerarsi un'azienda di medio/piccole dimensioni, di giovane età, in veloce espansione ed inserimento nel mercato digitale attraverso le sue numerose soluzioni home banking realizzate. Il progetto sviluppato e, in un futuro venduto, è caratterizzato da dimensioni e costi contenuti. Da tali informazioni si deduce che:

- il periodo di tirocinio, quindi il tempo dedicato al progetto, e le dimensioni medio/piccole, con la relativa giovinezza dell'azienda, comportino a prendere scelte rapide e sicure sull'uso delle tecnologie così da prediligere l'implementazione rapida di uno o più prototipi su cui sviluppare il prodotto finale;
- le tendenze tecnologiche del mercato sono fortemente dinamiche, necessitando così l'esigenza da parte dell'azienda nel volersi proporre con idee nuove ed innovative.

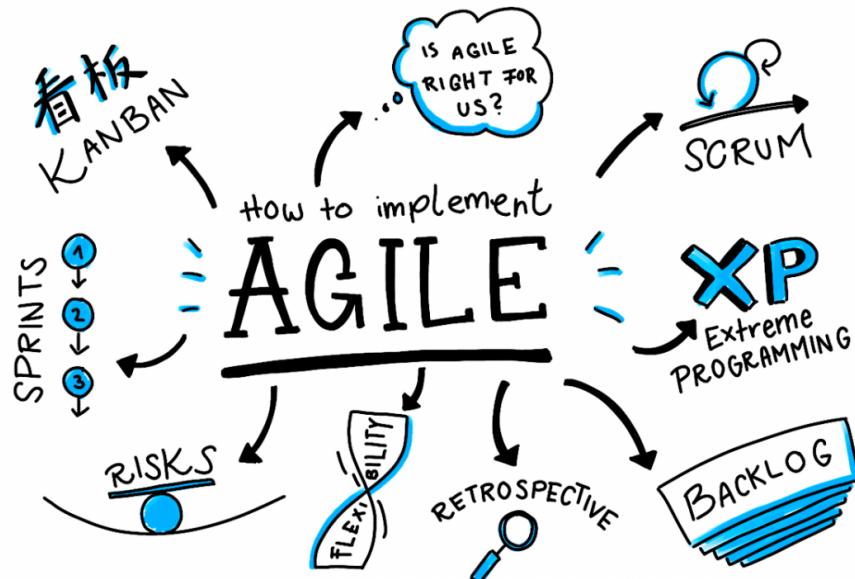


Figura 1.3: Modello Agile - SCRUM

Sulla base di tali considerazioni, in accordo con l’azienda, la scelta sulla metodologia di sviluppo è ricaduta sul modello di lavoro agile⁷ utilizzando il framework Scrum⁸. Tale framework viene adattato dell’azienda nel seguente modo:

- Team di sviluppo: ogni progetto viene assegnato a più macro-team di sviluppo, composto mediamente da cinque o sei persone, come per esempio esiste il team per lo sviluppo del backend, per la realizzazione del frontend ed altro. Nel caso del progetto Consierge Croccante è stato organizzato un micro-team di due persone per il backend e una sola persona del reparto grafica per il frontend;
- Sviluppo per incremento: l’azienda Crispy Bacon adotta appieno questo approccio di sviluppo, creando di volta in volta liste di funzionalità da implementare per l’incremento successivo;
- Scrum: la riunione che rappresenta lo Scrum viene effettuata ogni tre giorni dal micro-team che si occupa del backend, in presenza del CFO e fondatore Damiano Buscemi;
- ScrumMaster: all’interno del micro-team il tutor aziendale ha ricoperto il ruolo di ScrumMaster;
- Sprint: il concetto di sprint definito nel framework Scrum viene applicato ed eseguito appieno dall’azienda. Uno sprint è stato disposto per la durata di circa due settimane;

⁷<https://agilemanifesto.org/iso/it/manifesto.html>

⁸Framework Scrum. URL: [https://it.wikipedia.org/wiki/Scrum_\(informatica\)](https://it.wikipedia.org/wiki/Scrum_(informatica))

1.4 Tecnologie utilizzate

1.4.1 Node.js

Per la realizzazione del codice è stato scelto di utilizzare come linguaggio Node.js, una runtime multiplataforma orientato agli eventi per l'esecuzione di codice JavaScript Server-side. Tale scelta è motivata dal fatto che i servizi implementati offrono API e pacchetti per questo linguaggio. Node.js consente di utilizzare JavaScript per scrivere codice da eseguire lato server, questo permette quindi di eseguire il codice sui servizi cloud computing quali Amazon Lambda. Altra particolarità di Node.js la sua architettura orientata agli eventi che rende possibile l'inserimento di input/output asincrono.

1.4.2 Alexa Presentation Language

Come accennato il progetto è costituito da un'interfaccia grafica che servirà di supporto al compimento del dialogo con la Skill. Per ottenere ciò è stato scelto di utilizzare Alexa Presentation Language⁹ messo a disposizione da Amazon. Esso permette di sviluppare il software con esperienze vocali e visive utilizzando layout. Si ha pertanto la flessibilità di sviluppare Skill multimodali con elementi visivi, tra cui grafica, immagini e la personalizzazione di output per dispositivi diversi. Quando si progetta utilizzando Alexa Presentation Language, vengono creati dei documenti APL, ovvero file JSON inviati dalla Skill al dispositivo. Quest'ultimo elabora il documento APL, importa immagini e dati per poi mostrare il risultato.

1.4.3 Servizi Amazon Web Service

AWS offre un ecosistema popolato da numerosissimi servizi già connessi tra loro. La scelta quindi di utilizzare tale ecosistema risulta la più conveniente e la più ottimale. Pertanto nel progetto vengono utilizzati i seguenti servizi:

- AWS IAM: per gestire la sicurezza e gli accesso ai servizi e risorse di AWS
- AWS Cloud Watch: per monitorare e osservare l'esecuzione delle funzioni
- AWS DynamoDB: per il servizio di database non relazionale
- AWS Lambda: per eseguire codice senza dover effettuare il provisioning e gestire server
- AWS SES: per il servizio di invio mail
- AWS S3: per il servizio di storage di oggetti

1.4.4 Google Calendar

Obiettivo importante del prodotto è quello di poter consultare un calendario dove vengono trascritti gli impegni, eventi e appuntamenti dell'azienda. La scelta è ricaduta su l'utilizzo di Google Calendar, già utilizzato da Crispy Bacon, ovvero un sistema di calendari realizzato da Google che offre la possibilità di creare più calendari, di condividerli e importarli da altri servizi online. Difatti, quest'ultima caratteristica, agevola il collegamento tra Skill e calendario grazie a pacchetti ed API documentate.

⁹ Alexa Presentation Language. URL: <https://developer.amazon.com/it/docs/alexa-presentation-language/apl-overview.html>

1.4.5 Slack

Per l'invio di notifiche la scelta ricade su Slack, uno strumento software che permette di inviare messaggi in modo istantaneo, già conosciuto ed utilizzato da Crispy Bacon. Tale software agevola il collegamento tra Skill e la notifica istantanea grazie ad API pubbliche e documentate.

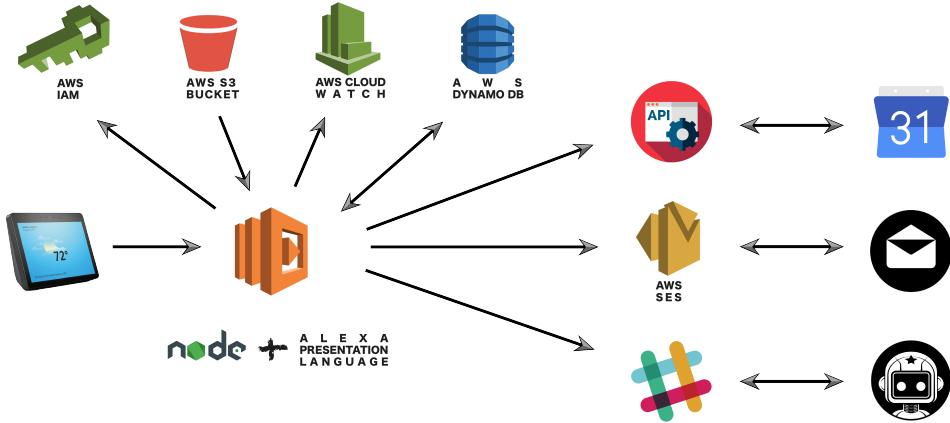


Figura 1.4: Architettura progetto C.C.

1.5 Strumenti utilizzati

1.5.1 Alexa Developer Console

Per realizzare la Skill è essenziale l'uso di Alexa Developer Console¹⁰, una console per gli sviluppatori che organizza lo sviluppo nelle seguenti attività principali:

- Build: per realizzare e impostare la Skill creata, configurare il modello di interazione e specificare gli endpoint per il servizio.
- Test: per testare le Skill con l'assistente Alexa usando testo o voce.
- Distribuzione: per vedere in anteprima come appariranno le Skill nello store.
- Certificazione: per convalidare la Skill creata, eseguire test di pre-certificazione e infine inviare la Skill per la certificazione.
- Analytics: per rivedere le metriche relative alle Skill create.

1.5.2 Amazon Echo Show

Per vedere e testare il prodotto finale si è usato l'Amazon Echo Show, uno smart display da 10" che integra l'assistente vocale Amazon Alexa. Controllabile con la voce, è in grado di interagire con gli utenti, eseguire diversi comandi vocali oltre che mostrare i contenuti direttamente sullo schermo touch.

¹⁰ Alexa Developer Console. URL: <https://developer.amazon.com/it/docs/devconsole/about-the-developer-console.html>

1.5.3 NPM

Per installare i pacchetti necessari è stato utilizzato npm, il principale software per gestire i moduli di Node.js e consente di condividere il codice per problemi tipici tra gli sviluppatori JavaScript. La filosofia alla base di tale gestore è quella che se un problema è stato già risolto da altri programmati deve esser possibile utilizzare la soluzione condivisa messa a disposizione degli altri utenti. npm oltre a consentire il riuso del codice, consente di tenerlo costantemente sotto controllo in modo da aggiornarlo. npm suddivide inoltre il codice in package, ovvero una directory che contiene uno o più file insieme, dove un file chiamato package.json contiene i dati relativi al pacchetto.

1.5.4 Gitlab

Il versionamento degli applicativi viene fatto tramite Git¹¹ e i repository sono ospitati su Gitlab¹² in un server dedicato dell'azienda. GitLab è una piattaforma che permette la gestione centralizzata dei repository Git, permettendo l'amministrazione dei permessi d'accesso tramite una semplice interfaccia grafica. Tutti gli applicativi e le configurazioni dell'infrastruttura vengono ospitati sulla piattaforma e il versionamento segue regole del Git-Flow dove le funzionalità nuove vengono sviluppate su un branch dedicato, per poi essere spostate successivamente nei branch di sviluppo, accettazione e infine produzione.

1.5.5 Ambiente di sviluppo

L'azienda non ha posto alcuna limitazione per quanto riguarda gli IDE lasciando libero arbitrio sulla scelta. Necessitando il progetto di diversi linguaggi quali Node.js e JSON la scelta è ricaduta su Visual Studio Code¹³, IDE sviluppato da Microsoft¹⁴ e consigliato per lavorare con progetti multi linguaggio, in quanto estensibile tramite plugin per supportare la maggior parte dei linguaggi di programmazione.

1.6 Organizzazione del testo

Il documento presenta la seguente struttura:

- Capitolo 1: Introduzione comprende la descrizione generale dell'azienda ospitante e del relativo metodo di lavoro, una contestualizzazione ed illustrazione del progetto di stage ed infine la presentazione della struttura del documento
- Capitolo 2: Analisi dei Requisiti si tratta di una descrizione dettagliata della fase di analisi dei requisiti portata a termine dal candidato
- Capitolo ??
- Capitolo ??
- Capitolo ??

¹¹Git. URL: <https://git-scm.com/>

¹²Gitlab. URL: <https://about.gitlab.com/>

¹³Visual Studio Code. URL: <https://code.visualstudio.com/>

¹⁴Microsoft. URL: <https://www.microsoft.com/>

Capitolo 2

Analisi dei Requisiti

Durante il primo periodo di tirocinio è stata svolta la fase di Analisi dei Requisiti, necessaria alla comprensione del dominio e al soddisfacimento della richiesta. Inizialmente è stato effettuato un incontro con il tutor aziendale con il quale sono stati rilevati i requisiti obbligatori richiesti dall'azienda da cui è stato possibile identificare un insieme di funzionalità necessarie alla Skill. Successivamente è stato realizzato il documento contenente tutti gli studi di analisi che hanno favorito una buona e corretta progettazione del prodotto. Dalla raccolta dei dati e dalle analisi fatte è stata elaborata una visione ad alto livello del sistema e dei rispettivi casi d'uso. Sono stati inoltre identificati quei punti critici in grado di determinare, in larga misura, la forma finale del prodotto. La fase di analisi si è infine conclusa con lo studio della VUI (Voice User Interface), e della GUI (Graphical User Interface), che caratterizzano l'esperienza d'uso del prodotto finale.

2.1 Obiettivo

L'obiettivo del progetto di stage è la realizzazione di una Skill per l'assistente vocale Alexa che sia in grado di accogliere clienti, postini e corrieri all'ingresso degli uffici e notificare in maniera automatica la persona interessata nell'arrivo del visitatore. La Skill è concepita per essere installata sui dispositivi in commercio da Amazon, in particolare sul dispositivo Echo Show 2018. L'obiettivo è quindi quello di realizzare un concierge virtuale che accolga il visitatore e riceva da esso informazioni per mezzo di una conversazione e l'uso di messaggi attraverso lo schermo touch integrato nel dispositivo. Infine tali dati elaborati dai processi di controllo per inviare notifiche al personale.

2.1.1 Cliente finale

Il cliente finale a cui è destinato il prodotto è l'azienda Crispy Bacon Srl, la quale necessita, per ovvi motivi, di un sistema automatizzato che svolga il ruolo di Concierge all'entrata degli uffici. L'azienda desidera, con questo prodotto, realizzare uno prototipo da poter rendere spendibile tale servizio proponendolo come pacchetto preconfezionato o da personalizzare al fine di venderlo a clienti terzi.

2.2 Utenti coinvolti

Dall'analisi fatta sono emersi i seguenti attori, intesi come persone coinvolte nell'utilizzo della Skill:

- Visitatore
 - Persona avente appuntamento
 - Persona non avente appuntamento
 - Postino
 - Corriere
- Personale di Crispy Bacon

2.3 Requisiti richiesti

I requisiti emersi e richiesti dall'azienda sono stati elaborati e analizzati. Tali requisiti sono interpretabili in tre diversi modi, tutti con il presupposto che questi siano in qualche modo una necessità.

- Requisito utente: dal punto di vista dell'utente, è una capacità necessaria per risolvere un problema o raggiungere un obiettivo.
- Requisito software: dal punto di vista della soluzione, è una capacità che deve essere posseduta dal sistema per adempiere all'obiettivo.
- Dal punto di vista della documentazione, come una descrizione documentata di una capacità interpretata come un requisito utente o software.

2.3.1 Identificazione dei Requisiti

Per identificare i requisiti viene utilizzata la seguente notazione tabellare con un codice che lo identifica e la corrispettiva descrizione a fianco.

R.x.y

- R: identifica il requisito
- x: identifica l'importanza di tale requisito che può essere
 - O obbligatorio
 - D desiderabile
- y: identifica un valore numerico progressivo a partire da 1

In merito allo studio di analisi fatto all'inizio del periodo di tirocinio sono emersi i seguenti requisiti, **considerati obbligatori** per il soddisfacimento del risultato atteso dal prodotto finale. I requisiti riportati vengono divisi per Voice User Interface, Graphical User Interface e funzionalità.

Requisiti analizzati per il Voice User Interface:

Identificativo	Requisito
RO1	La Skill al momento del lancio deve presentare un messaggio di benvenuto (VUI)
RO2	Successivamente al lancio della Skill, essa deve presentare un elenco essenziale e sintetico delle azioni disponibili (VUI)
RO3	La Skill deve poter ricevere le informazioni necessarie dal visitatore per mezzo di una conversazione impostata (VUI)
RO4	La Skill al termine del dialogo deve restituire una risposta appropriata nel caso di controlli positivi (VUI)
RO5	La Skill al termine del dialogo deve restituire una risposta appropriata nel caso di controlli negativi (VUI)
RO6	La Skill deve porre almeno 2 volte la domanda, posta in maniera differente dalla precedente, nel caso non riceva alcun comando dall'utente (VUI)
RO7	La Skill deve porre almeno 2 volte la domanda, posta in maniera differente dalla precedente, nel caso la risposta ricevuta attesa non dovesse essere corretta. (VUI)
RO8	La Skill deve notificare al visitatore l'arrivo del dipendente cercato (VUI).

Tabella 2.1: Tabella tracciamento requisiti obbligatori - VUI

Requisiti analizzati per il Graphical User Interface:

Identificativo	Requisito
RO9	La Skill al momento del lancio deve presentare un messaggio di benvenuto (GUI)
RO10	Successivamente al lancio della Skill, essa deve presentare un elenco essenziale e sintetico delle azioni da fare (GUI)
RO11	La Skill deve riportare il risultato finale del dialogo nello schermo del dispositivo (GUI)
RO12	La Skill deve, qualora necessario, presentare una lista di nomi dei dipendenti dell'azienda (GUI)
RO13	La lista citata nel requisito RO11 deve poter essere cliccabile nello schermo touch del dispositivo (GUI)
RO14	La Skill poter entrare in modalità presentazione per mezzo di un comando vocale (GUI)
RO15	La Skill consigliare i comandi a video (GUI)

Identificativo	Requisito
RO16	La Skill deve notificare al visitatore l'arrivo del dipendente cercato (GUI).

Tabella 2.2: Tabella tracciamento requisiti obbligatori - GUI

Requisiti analizzati per funzionalità:

Identificativo	Requisito
RO17	La Skill deve poter utilizzare e interrogare il servizio di calendariizzazione di Google con le informazioni ottenute dal visitatore al fine di notificare l'interessato della visita.
RO18	La Skill deve inviare una notifica all'interessato della visita.
RO19	La Skill deve inviare una notifica all'interessato al momento di una consegna di un pacco e/o lettera.

Tabella 2.3: Tabella tracciamento requisiti obbligatori - Funzionalità

Infine nell'analisi sono stati individuati anche i **requisiti desiderabili**, considerati non strettamente necessari ma di valore aggiunto al prodotto atteso.

Identificativo	Requisito
RD1	Rifacendosi al requisito RO1, la Skill al momento del lancio deve presentare un messaggio di benvenuto diverso da quello precedente (VUI)
RD2	La Skill una volta verificata la presenza del visitatore informa quest'ultimo se l'interessato risulta non reperibile se assente
RD3	La Skill deve poter registrare il momento in cui il visitatore inizia l'incontro con la persona cercata nel caso di un appuntamento
RD4	La Skill deve poter registrare il momento in cui il visitatore termina l'incontro con la persona cercata nel caso di un appuntamento

Tabella 2.4: Tabella tracciamento requisiti desiderabili

2.4 Casi d'uso

Dalle analisi fatte e dai dati raccolti sono stati studiati i requisiti funzionali, ovvero i casi d'uso del prodotto Concierge Crocante, che permettono di descrivere interazioni tra gli utenti, tra il sistema e come quest'ultimo deve essere utilizzato. Durante lo stage sono state così esaminate le sequenze di passi che descrivono interazioni e la rappresentazione di possibilità, che hanno in comune uno scopo finale per un utente (attore).

2.4.1 Attori

Gli attori emersi nell'analisi dei casi d'uso svolgono il ruolo dell'utente nell'interazione con la Skill per raggiungere l'obiettivo prefissato. In questa analisi sono stati individuati gli attori:

- Utente generico, che può essere generalizzato in:
 - Visitatore
 - * aventure appuntamento
 - * non aventure appuntamento
 - * postino
 - * corriere
 - Persona cercata

Di seguito viene riportato lo schema degli attori individuati utilizzando lo standard UML 2.0

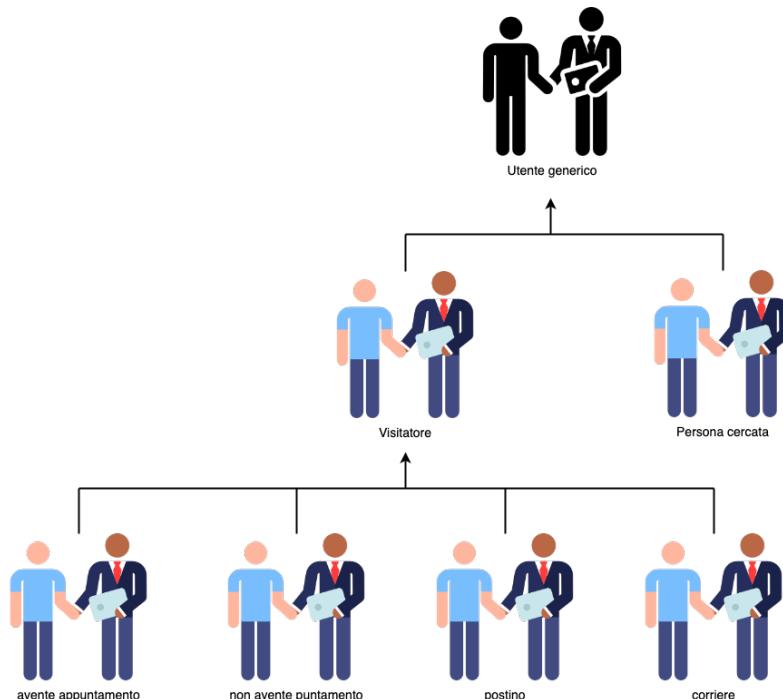


Figura 2.1: Utenti del sistema

2.4.2 Casi d'uso - Visitatore

Durante il periodo di tirocinio, la fase di analisi dei casi d'uso è da considerarsi divisa in due parti simili fra loro: quella dedicata al visitatore, ovvero colui che si presenta negli uffici dell'azienda avente un appuntamento registrato in calendario, o che semplicemente si presenta senza preavviso, e il servizio di consegna pacchi svolto dal postino o dal corriere. In entrambe le parti gli attori avvieranno la Skill installata nel dispositivo Amazon Echo Show e seguiranno le istruzioni riportate a voce dall'assistente vocale oppure mostrate a video dal display. In questa sezione viene riportata una rappresentazione grafica dei casi d'uso dell'utente identificato come "Visitatore avente appuntamento", che descrive le interazioni che l'attore svolge con il sistema. Di seguito viene riportato il diagramma dei casi d'uso individuati utilizzando lo standard UML 2.0

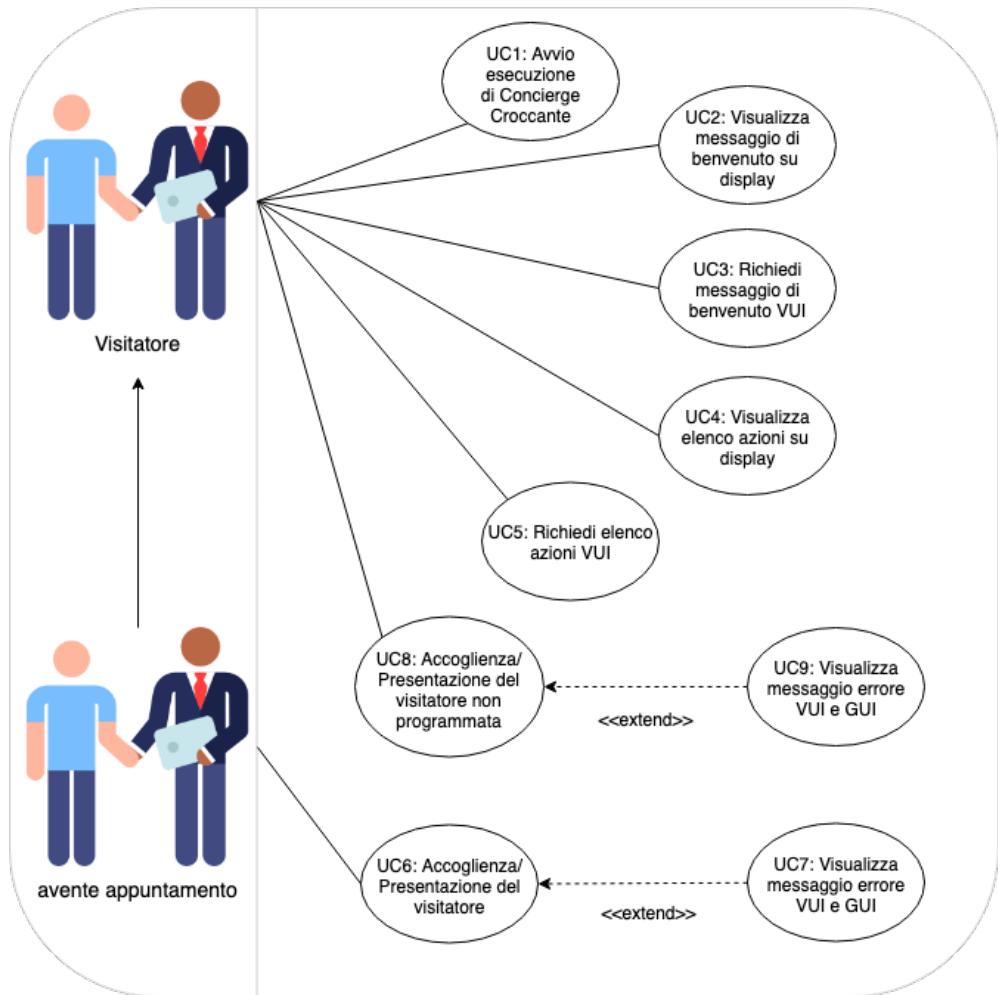


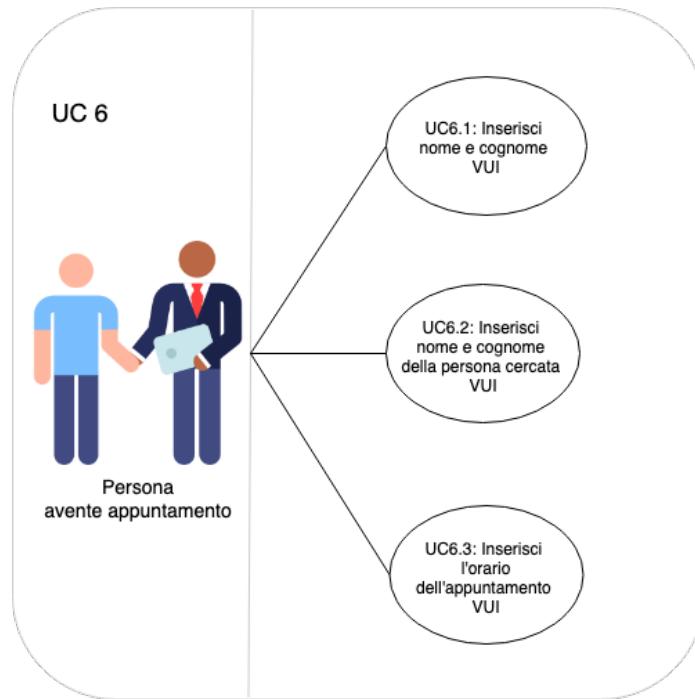
Figura 2.2: Casi d'uso visitatore avente appuntamento

Dall'elaborazione di tale analisi sono quindi emersi i seguenti casi d'uso riportati:

Identificativo	Descrizione
UC1	<p><i>Attori:</i> visitatore avente o non avente appuntamento</p> <p><i>Scopo:</i> l'utente può avviare Concierge Croccante</p> <p><i>Pre-condizione:</i> il dispositivo Amazon deve aver avviato l'assistente Alexa</p> <p><i>Post-condizione:</i> l'utente ha avviato Concierge Croccante</p>
UC2	<p><i>Attori:</i> visitatore</p> <p><i>Scopo:</i> l'utente riceve un messaggio di benvenuto sul display del dispositivo</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> l'utente riceve un messaggio di benvenuto sul display del dispositivo</p>
UC3	<p><i>Attori:</i> visitatore</p> <p><i>Scopo:</i> l'utente riceve un messaggio vocale di benvenuto dalla Skill Concierge Croccante</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> l'utente riceve un messaggio vocale dalla Skill Concierge Croccante</p>
UC4	<p><i>Attori:</i> visitatore</p> <p><i>Scopo:</i> l'utente visualizza l'elenco sintetico ed essenziale di azioni sul display del dispositivo</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> l'utente visualizza l'elenco di azioni sul display del dispositivo</p>
UC5	<p><i>Attori:</i> visitatore</p> <p><i>Scopo:</i> viene esposto all'utente l'elenco sintetico ed essenziale di azioni</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> viene esposto all'utente l'elenco di azioni disponibili da Concierge Croccante</p>
UC6	<p><i>Attori:</i> visitatore avente appuntamento</p> <p><i>Scopo:</i> l'utente può annunciarsi per essere accolto dalla persona cercata</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> l'utente si annuncia</p>

Identificativo	Descrizione
UC7	<p><i>Attori:</i> visitatore avente appuntamento</p> <p><i>Scopo:</i> viene visualizzato un messaggio, su display e vocale, di errore specifico per l'eccezione riscontrata</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> l'utente riceve un messaggio su display e vocale di errore</p>
UC8	<p><i>Attori:</i> visitatore</p> <p><i>Scopo:</i> l'utente può annunciarsi per essere accolto dalla persona cercata</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> l'utente si annuncia</p>
UC9	<p><i>Attori:</i> visitatore</p> <p><i>Scopo:</i> viene visualizzato un messaggio, su display e vocale, di errore specifico per l'eccezione riscontrata</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> l'utente riceve un messaggio su display e vocale di errore</p>

Tabella 2.5: Tabella casi d'uso visitatore avente appuntamento e non

**Figura 2.3:** Sotto casi d'uso visitatore avente appuntamento

Identificativo	Descrizione
UC6.1	<p><i>Attori:</i> visitatore avente o non avente appuntamento <i>Scopo:</i> l'utente inserisce/dice il suo nome e cognome <i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata <i>Post-condizione:</i> l'utente ha inserito/detto il suo nome e cognome</p>
UC6.2	<p><i>Attori:</i> visitatore avente o non avente appuntamento <i>Scopo:</i> l'utente inserisce/dice il nome e cognome della persona che cerca <i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata <i>Post-condizione:</i> l'utente ha inserito/detto il nome e cognome della persona cercata</p>
UC6.3	<p><i>Attori:</i> visitatore avente o non avente appuntamento <i>Scopo:</i> l'utente inserisce/dice l'orario di appuntamento <i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata <i>Post-condizione:</i> l'utente ha inserito/detto l'orario di appuntamento</p>

Tabella 2.6: Tabella sotto casi d'uso visitatore avente appuntamento e non

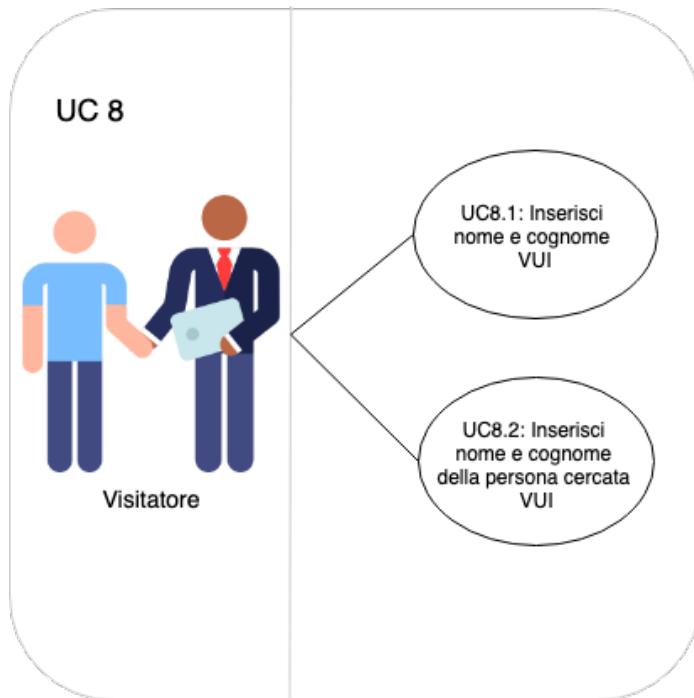


Figura 2.4: Sotto casi d'uso visitatore avente appuntamento

Identificativo	Descrizione
UC8.1	<p><i>Attori:</i> visitatore</p> <p><i>Scopo:</i> l'utente inserisce/dice il suo nome e cognome</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> l'utente ha inserito/detto il suo nome e cognome</p>
UC8.2	<p><i>Attori:</i> visitatore</p> <p><i>Scopo:</i> l'utente inserisce/dice il nome e cognome della persona che cerca</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> l'utente ha inserito/detto il nome e cognome della persona cercata</p>

Tabella 2.7: Tabella sotto casi d'uso visitatore avente appuntamento e non

2.4.3 Casi d'uso - Postino/Corriere

La seconda parte di analisi dei casi d'uso, è dedicata al servizio di consegna dei pacchi che viene svolto dal postino o dal corriere. In questa sezione si rappresenta graficamente i casi d'uso dell'utente identificato come "Corriere", che descrive le interazioni che l'attore svolge con il sistema.

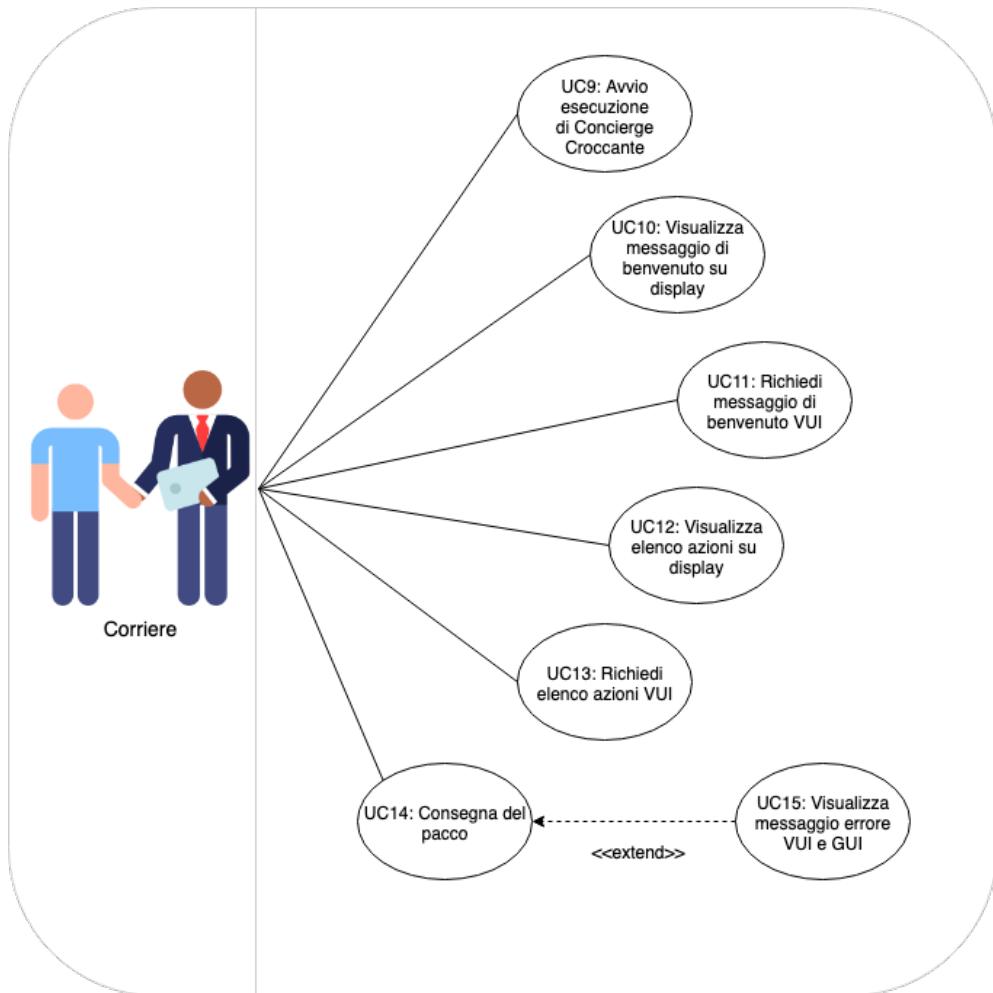


Figura 2.5: Casi d'uso corriere

Dall'elaborazione di tale analisi sono quindi emersi i seguenti casi d'uso riportati:

Identificativo	Descrizione
UC9	<p><i>Attori:</i> corriere</p> <p><i>Scopo:</i> l'utente può avviare Concierge Croccante</p> <p><i>Pre-condizione:</i> il dispositivo Amazon deve aver avviato l'assistente Alexa</p> <p><i>Post-condizione:</i> l'utente ha avviato Concierge Croccante</p>
UC10	<p><i>Attori:</i> corriere</p> <p><i>Scopo:</i> l'utente riceve un messaggio di benvenuto sul display del dispositivo</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> l'utente riceve un messaggio di benvenuto sul display del dispositivo</p>
UC11	<p><i>Attori:</i> corriere</p> <p><i>Scopo:</i> l'utente riceve un messaggio vocale di benvenuto dalla Skill Concierge Croccante</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> l'utente riceve un messaggio vocale dalla Skill C.C.</p>
UC12	<p><i>Attori:</i> corriere</p> <p><i>Scopo:</i> l'utente visualizza l'elenco sintetico ed essenziale di azioni sul display del dispositivo</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> l'utente visualizza l'elenco di azioni sul display del dispositivo</p>
UC13	<p><i>Attori:</i> corriere</p> <p><i>Scopo:</i> viene esposto all'utente l'elenco sintetico ed essenziale di azioni</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> viene esposto all'utente l'elenco di azioni disponibili da C.C.</p>
UC14	<p><i>Attori:</i> corriere</p> <p><i>Scopo:</i> l'utente può consegnare un pacco</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> l'utente consegna il pacco</p>

Identificativo	Descrizione
UC15	<p><i>Attori:</i> corriere</p> <p><i>Scopo:</i> viene visualizzato un messaggio, su display e vocale, di errore specifico per l'eccezione riscontrata</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> l'utente riceve un messaggio su display e vocale di errore</p>
UC17	<p><i>Attori:</i> corriere</p> <p><i>Scopo:</i> l'utente per consegnare il pacco richiede la presenza di una persona specifica per una firma</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> l'utente riceve la firma desiderata e consegna il pacco</p>
UC18	<p><i>Attori:</i> corriere</p> <p><i>Scopo:</i> l'utente per consegnare il pacco richiede la presenza di una persona per una firma</p> <p><i>Pre-condizione:</i> la Skill Concierge Croccante deve essere stata avviata</p> <p><i>Post-condizione:</i> l'utente riceve la firma desiderata e consegna il pacco</p>

Tabella 2.8: Tabella casi d'uso corriere

2.5 Voice User Interface - VUI

Altro aspetto importante dell'analisi è stata sulla Voice User Interface, che nel documento verrà abbreviato con l'acronimo VUI. La VUI per la Skill è l'interfaccia che rende possibile l'interazione umana parlata con il computer, utilizzando il riconoscimento vocale per comprendere comandi e domande vocali, ed infine il text to speech per riprodurre una risposta. Nel caso del progetto la VUI è rappresentata dall'assistente vocale Amazon Alexa, la quale eseguirà la Skill prodotta. Vista la ovvia complessità che presenta un'interazione umana vocale con un computer, la VUI fornita dall'assistente vocale Alexa presenta alcuni limiti che verranno analizzati successivamente in un nuovo punto del capitolo.

L'obiettivo che si prefigge nel realizzare la Skill è quello di poter sostenere una conversazione, il più naturale possibile, con la persona accolta all'entrata dell'azienda Crispy Bacon. L'idea è quindi che l'ospite possa dialogare con Concierge Croccante, motivare la presenza in azienda e notificare l'interessato della visita. Dall'analisi emerge una conversazione come l'esempio seguente in formato testo di quello che si vuole ottenere dal prodotto finale:



Figura 2.6: Esempio di conversazione

- **Utente:** "Alexa, sono qui."
- **Concierge:** "Buongiorno! Benvenuto a Crispy Bacon, quale è il motivo della sua visita?"
- **Utente:** "Devo consegnare un pacco."
- **Concierge:** "Chi è il destinatario del pacco?"
- **Utente:** "Matteo P."
- **Concierge:** "Ho notificato a Matteo che c'è un pacco per lui, sta per arrivare."
- **Concierge:** "Crispy Bacon la ringrazia"

La Skill Concierge Croccante necessita quindi di ricevere dati in input per poter completare il processo di accoglienza della persona entrata in azienda. È quindi necessario suddividere i dati richiesti in base alla tipo di utente:

Tipo di utente	Dati da raccogliere
Persona senza appuntamento	<ul style="list-style-type: none"> - Proprio nome - Proprio cognome - Nome persona cercata - Cognome persona cercata
Persona con appuntamento	<ul style="list-style-type: none"> - Proprio nome - Proprio cognome - Nome persona cercata - Cognome persona cercata - Orario dell'appuntamento
Corriere	<ul style="list-style-type: none"> - Il proprio titolo (corriere) - Nome della persona interessata - Cognome della persona interessata
Postino	<ul style="list-style-type: none"> - Il proprio titolo (postino) - Nome della persona interessata - Cognome della persona interessata

Tabella 2.9: Tabella tipologia dati per tipo di utente

2.5.1 Voice Flow

Al termine della fase di analisi sulla VUI si ha ottenuto come risultato il Voice Flow, ovvero il percorso ipotetico che l'utente può intraprendere durante la conversazione con la Skill Concierge Croccante, rappresentato dal diagramma di attività secondo lo standard UML 2.0. Quest'ultimo è un tipo di diagramma che permette la descrizione un processo attraverso dei grafi in cui i nodi rappresentano le attività e gli archi l'ordine con cui vengono eseguite. Permette inoltre la descrivere gli aspetti dinamici dei casi d'uso e supporta l'elaborazione parallela.

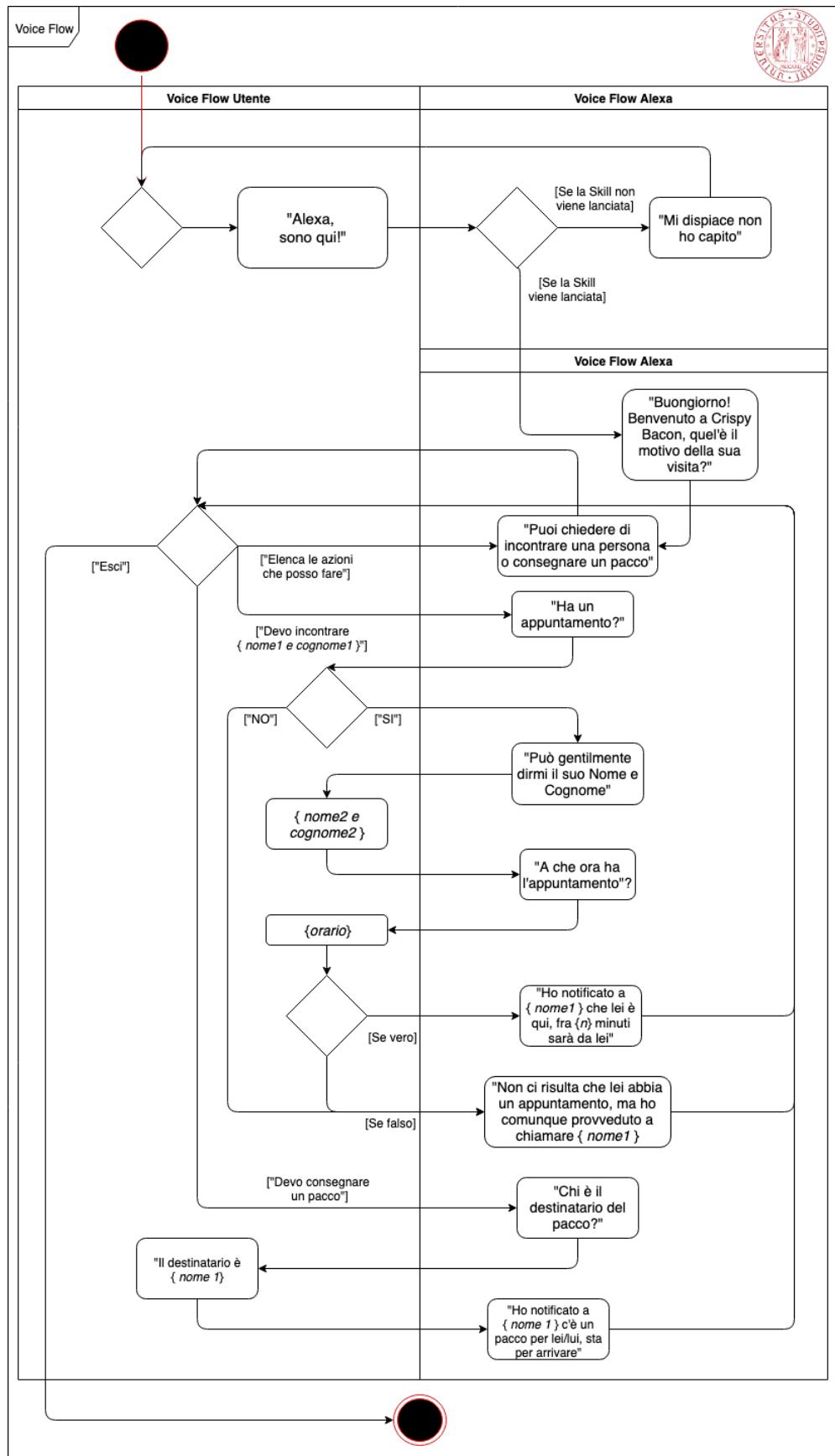


Figura 2.7: Diagramma di attività VUI - Concierge Croccante

2.5.2 Interazione vocale - UC6

Di seguito si riporta l'analisi descrittiva e testuale dell'interazione vocale del UC6, riportato in tabella ?, considerato significativo per la comprensione dei dialoghi.

- *Utente - Visitatore:* "Salve! Devo incontrare {nome1 e cognome1}."
- *Alexa - Concierge Croccante:* "Può gentilmente dirmi il suo nome e cognome?"
- *Utente - Visitatore:* "{nome2 e cognome2}."
- *Alexa - Concierge Croccante:* "A che ora ha l'appuntamento?"
- *Utente - Visitatore:* "{orario}."
- *Alexa - Concierge Croccante:* "Ho notificato a {nome1} che lei è qui, fra pochi secondi sarà da lei."

Altra casistica:

- *Utente - Visitatore:* "Salve! Sono {nome2 e cognome2}, devo incontrare {nome1 e cognome1}."
- *Alexa - Concierge Croccante:* "Ha un appuntamento?"
- *Utente - Visitatore:* "Sì."
- *Alexa - Concierge Croccante:* "A che ora ha l'appuntamento?"
- *Utente - Visitatore:* "{orario}."
- *Alexa - Concierge Croccante:* "Ho notificato a {nome1} che lei è qui, fra pochi secondi sarà da lei."

Nel caso in cui l'utente non abbia un appuntamento o non risulta nel calendario l'incontro, si riceverà la seguente risposta:

- *Alexa - Concierge Croccante:* "Non ci risulta che lei abbia un appuntamento, ma ho comunque provveduto a chiamare {nome1}."

2.6 Graphical User Interface - GUI

Un altro aspetto, studiato per concludere tutti gli aspetti dell'analisi, è stata la Graphical User Interface, che nel documento verrà abbreviato con l'acronimo GUI. La GUI è l'interfaccia grafica utente che consente l'interazione uomo-macchina in modo visuale utilizzando rappresentazioni grafiche. Nel caso del progetto la GUI è costituita da delle Multimodal Displays, ovvero delle visuals che riportano dei dati (anche linkati) e delle funzionalità, mostrate sullo schermo del dispositivo Amazon Echo Show. Le visuals mostrate sono quindi a supporto della VUI e hanno il compito di riportare a video informazioni utili a migliorare la conversazione uomo-macchina.



Figura 2.8: Esempio GUI - Home

La GUI svolge quindi il compito di supporto per aiutare e migliorare l'esperienza di dialogo. L'immagine sopra riporta una visual che propone di presentare dei dati secondo quel determinato layout. In questa fase di analisi vengono esposti alcuni layout significativi che mostrano a schermo alcune informazioni:



- Titolo - Nome dell'azienda
- Azione n
- Immagine azione n
- *altre informazioni*

Figura 2.9: Esempio di visual - Aiuto



- Titolo - Nome dell'azienda
- Informazione
- Immagine allegata
- *altre informazioni*

Figura 2.10: Esempio di visual - Consegna



- Titolo - Nome dell'azienda
- Informazioni sull'appuntamento
- Immagine allegata
- *altre informazioni*

Figura 2.11: Esempio di visual - Visitatore aente appuntamento

2.7 Limiti

Come già in precedenza menzionato gli assistenti vocali presentano dei limiti, in quanto non riescono a sostenere una conversazione completamente naturale con una persona umana. Nel contesto del progetto i limiti che si riscontrano nella Skill e Alexa sono i seguenti:

- Alexa
 - Non è possibile richiamare l'attenzione dell'assistente vocale senza pronunciare la parola "*Alexa*"
 - Per motivi linguistici l'assistente vocale può fare fatica a comprendere i nomi di persone
- Skill Concierge Croccante
 - La Skill non può essere avviata senza il lancio da parte di Alexa
 - La Skill, per motivi implementativi, non può rimanere in ascolto per più di 8 secondi in attesa di una risposta da parte dell'utente

Capitolo 3

Progettazione e codifica

In questo capitolo verrà trattato il secondo periodo del tirocinio: la progettazione e la codifica. Questa parte, avvenuta dopo l'Analisi dei Requisiti, è da ritenersi importante per progetto in quanto ha occupato gran parte del tempo lavorativo. Il primo passo è stato eseguire la progettata dell'architettura, in modo che i servizi necessari fossero correttamente configurati prima del loro effettivo utilizzo. Successivamente è stata eseguita la stesura del codice per la realizzazione della Skill. Nei paragrafi successivi seguiranno porzioni di codice significativo per la loro importanza e per il funzionamento di determinate funzionalità e/o servizi. Importante ricordare che il codice implementato e mostrato sarà in Node.js come quanto detto nel paragrafo [1.4.1](#).

3.1 Architettura

Come riportato nel paragrafo [1.4.3](#), il progetto si basa interamente sui servizi offerti dall'ecosistema Amazon. Di conseguenza è risultato facile ed immediato impostare i servizi dell'architettura in modo da rendere il tutto efficacie ed efficiente. In questo paragrafo si andrà a comprendere cosa offre ogni singolo servizio di AWS e come esso è stato impostato per il suo corretto funzionamento nella Skill.

3.2 Configurazione servizi Amazon Web Service

3.2.1 AWS SES

Amazon SES¹ (Simple Email Service) è un servizio di invio e-mail basato sul cloud messo a disposizione da Amazon per gli sviluppatori di applicazioni. Tale servizio è da considerarsi vantaggioso in quanto è affidabile e a costo ridotto, utile per qualunque tipo di azienda ed ideale per la realizzazione del progetto.

¹AWS SES. URL: <https://aws.amazon.com/it/ses/>

Per utilizzare AWS SES è necessario impostare il servizio visitando la pagina² dedicata ed eseguire le istruzioni riportate:



Figura 3.1: Icona AWS SES

- Eseguire l'accesso con le proprie credenziali di account AWS;
- Una volta fatto l'accesso cliccare **Email Addresses** sul pannello a sinistra;
- Cliccare **Verify a New Email Address**;
- Inserire l'indirizzo e-mail da verificare il quale verrà utilizzato per inviare le notifiche;
- Confermare la verifica cliccando sul URL contenuto nella e-mail ricevuta.

Il procedimento descritto sopra non fa altro che verificare l'indirizzo e-mail con il quale si andrà ad inviare messaggi di posta elettronica tramite la Skill. Amazon infatti permette l'uso di questo servizio solo se il mittente e il destinatario delle e-mail sono indirizzi verificati. Per completare la configurazione è necessario di verificare il dominio delle caselle mail di Crispy Bacon:

- Sempre all'interno di AWS SES cliccare su **Domains** sul pannello a sinistra;
- Cliccare in altro **Verify a New Domains**;
- Inserire il dominio degli indirizzi e-mail destinatari delle mail, nel caso del progetto *crispybacon.store*;
- Infine cliccare su **Verify This Domain** per terminare il processo di verifica.

A questo punto è possibile mandare messaggi e notifiche tramite mail utilizzando l'indirizzo verificato prima.

²AWS SES. URL: <https://aws.amazon.com/it/ses/>

3.2.2 AWS S3

Amazon S3³ (S3 = Simple Storage Service) è un servizio di storage di oggetti che offre scalabilità, disponibilità dati, sicurezza e prestazioni all'avanguardia. Queste caratteristiche offre alle industrie di qualsiasi dimensione la possibilità di archiviare e proteggere una qualsiasi quantità di dati per qualunque genere di uso: come ad esempio per siti Web, applicazioni mobile, backup e ripristino, archiviazione, applicazioni enterprise⁴, dispositivi IoT⁵ e analisi di big data. Amazon S3 offre una gestione semplice di utilizzo grazie alla sua pagina web dedicata, all'interno di AWS Console, che di organizzare i dati e di configurare controlli di accesso. S3 vanta di avere clienti di grande notorietà come Netflix, Airbnb, Finra e altri ancora. Nel caso del progetto S3 è stato utilizzato per organizzare e archiviare file multimediali, per lo più immagini, necessari per la composizione delle schermate APL mostrate a video sul display del dispositivo utilizzato. Per caricare tali file è necessario seguire le seguenti istruzioni riportate:

- Effettuare l'accesso con le proprie credenziali di account AWS;
- Una volta entrati nella pagina di S3 cliccare su **Crea bucket^a** sul pannello di opzioni riportato in alto;
- Apparirà una nuova finestra dove sarà necessario inserire il nome del bucket, nel caso del progetto *concierge-corccante*, e la regione del server che ospiterà i dati, in questo caso *UE Irlanda*. Infine cliccare su **Successivo** fino al termine della schermata lasciando invariate le impostazioni proposte;
- A questo punto il bucket (contenitore) è stato creato. Per entrare basterà ora cliccare sul suo nome nella lista dei bucket.

^aUn contenitore di oggetti memorizzato al suo interno

Il passo finale di questa configurazione sarà caricare i file multimediali che le schermate APL necessitano. Quindi una volta entrati nel bucket creato in precedenza:

- Cliccare sul **Carica** sul pannello di opzioni riportato in alto;
- Apparirà una nuova finestra dove aggiungere uno più file cliccando su **Aggiungi**;
- Per procedere cliccare su **Successivo** e alla seconda schermata impostare le autorizzazioni pubbliche su *Concedi l'accesso pubblico..*;
- Continuare cliccando su **Successivo** fino al termine della schermata lasciando invariate le impostazioni proposte.

Ora i file sono archiviati e organizzati, pronti per essere utilizzati dalle schermate APL.

³AWS S3. URL: <https://aws.amazon.com/it/s3/>

⁴Integrazione tra diversi tipi di sistemi informatici con l'utilizzo di software e soluzioni architetturali

⁵Internet of Things: neologismo utilizzato per dare un nome agli oggetti reali connessi ad internet



Figura 3.2: Icona AWS S3

3.2.3 AWS IAM

Amazon IAM⁶ (Identity and Access Management) consente di gestire in sicurezza l'accesso ai servizi e alle risorse di AWS. Con questo servizio di management è possibile creare ed amministrare utenti e gruppi di utenti in modo da autorizzare o negare loro l'accesso alle risorse di AWS. Per ovvie ragioni, IAM svolge un ruolo importante nel progetto in quanto abilita e disabilita l'uso di tutti i servizi di Amazon usati dalla Skill. È pertanto necessario porre particolare attenzione ai passaggi di configurazione del gestore di sicurezza così da garantire il corretto funzionamento del prodotto finale:



Figura 3.3: Icona AWS IAM

- Eseguire l'accesso con le proprie credenziali di account AWS;
- Una volta entrati bisognerà recarsi su Policy presente nel pannello a sinistra e cliccare su **Crea policy**;
- Si verrà indirizzati in una nuova pagina dove sarà necessario andare nella sezione JSON ed incollare il codice sotto riportato utilizzato per il progetto;
- Infine cliccare su **Verifica policy**.

```

1   {
2       "Version": "2012-10-17",
3       "Statement": [
4           {
5               "Effect": "Allow",
6               "Action": [
7                   "dynamodb:BatchGetItem",
8                   "dynamodb:GetItem",
9                   "dynamodb:Query",
10                  "dynamodb:Scan",
11                  "dynamodb:BatchWriteItem",
12                  "dynamodb:PutItem",
13                  "dynamodb:UpdateItem"
14             ], "Resource": "*"
15         },
16         {
17             "Effect": "Allow",
18             "Action": [ "logs:CreateLogStream", "logs:PutLogEvents" ],
19             "Resource": "arn:aws:logs:eu-west-1:679871574158:*log"
20         },
21         {
22             "Effect": "Allow",
23             "Action": "logs>CreateLogGroup",
24             "Resource": "*"
25         },
26         {
27             "Effect": "Allow",
28             "Action": [ "ses:*" ],
29             "Resource": "*"
29     ]
29 }

```

Listing 3.1: Esempio Policy IAM

Ora è necessario associare le policy create con le roles:

⁶AWS IAM. URL: <https://aws.amazon.com/it/iam/>

- Quindi andare sulla sezione Ruoli presente nel pannello a sinistra e cliccare su **Crea ruolo**;
- Nella nuova pagina selezionare il servizio che utilizzerà il nuovo ruolo, quindi scegliere **Lambda** e cliccare su **Successivo**;
- La pagina successiva mostra l'elenco di policy disponibili, quindi scegliere la policy creata precedentemente e continuare cliccando su **Successivo**;
- Completare la procedura seguendo le istruzioni della pagina;
- Terminato tale processo sarà ora possibile associare la role creata con la Lambda, che verrà fatto al momento della creazione, per permettere alla funzione di utilizzare i servizi AWS necessari.

Ruoli > ConciergeCroccanteRole

Riepilogo

ARN ruolo	arn:aws:iam::679871574158:role/ConciergeCroccanteRole		
Descrizione ruolo	Abilita la Lambda functions a chiamare i servizi AWS. Modifica		
ARN del profilo dell'istanza			
Percorso	/		
Data di creazione	2019-06-20 16:32 UTC+0200		
Durata massima delle sessioni CLI/API	1 ora Modifica		
Autorizzazioni Relazioni di attendibilità Tag Consulente accessi Revoca sessioni			
▼ Policy di autorizzazioni (1 policy applicata)			
Collega policy			
<table border="1"> <thead> <tr> <th>Nome policy</th> </tr> </thead> <tbody> <tr> <td>ConciergeCroccantePolicy</td> </tr> </tbody> </table>		Nome policy	ConciergeCroccantePolicy
Nome policy			
ConciergeCroccantePolicy			

Figura 3.4: Esempio di Role creata

3.2.4 AWS CloudWatch

Amazon CloudWatch⁷ è un servizio di monitoraggio pensato per gli sviluppatori per fornire dati e analisi dal monitoraggio delle applicazioni, così da rispondere ai cambiamenti di prestazioni a livello di sistema, ottimizzare l'utilizzo delle risorse e ottenere una visualizzazione unificata dello stato di integrità operativa. CloudWatch raccoglie i dati di monitoraggio sotto forma di log, parametri ed eventi, unificando la loro visualizzazione, sulle applicazioni e i servizi eseguiti in AWS. Nel caso del progetto Concierge Croccante CloudWatch è stato utilizzato per rilevare comportamenti anomali della Skill, così da rilevare gli eventuali errori visualizzandone i log.



Figura 3.5: Icona AWS CloudWatch

Nel punto precedente, ovvero nel paragrafo 3.2.3, è stato eseguito il procedimento necessario perché il servizio di CloudWatch venga abilitato. Pertanto non è necessario alcuna configurazione. Basterà recarsi al sito per poter utilizzare il servizio qualora si voglia visualizzare i dati sotto forma di log dopo o durante l'esecuzione di un servizio AWS, nel caso del progetto il servizio di AWS Lambda.

3.2.5 AWS DynamoDB

Amazon DynamoDB⁸ è un servizio di database NoSQL (database non relazionale) proprietario e completamente gestito che supporta strutture di dati di tipo documento e di tipo chiave-valore. Caratteristiche di pregio sono database durevoli, multiregione e offrono sicurezza, backup, ripristino e cache in memoria per le applicazioni Internet. DynamoDB può gestire oltre 10 trilioni di richieste al giorno e supportando picchi di oltre 20 milioni di richieste al secondo. Nel progetto La Skill realizzata utilizza tale servizio per interrogare un database contenente la lista dei nomi del personale Crispy Bacon.

⁷ AWS CloudWatch. URL: <https://aws.amazon.com/it/cloudwatch/>

⁸ AWS DynamoDB. URL: <https://aws.amazon.com/it/dynamodb/>



Figura 3.6: Icona AWS DynamoDB

È necessario quindi impostare tale servizio. Per farlo occorre recarsi nella pagina dedicata del db e una volta eseguito l'accesso con le proprie credenziali seguire le seguenti istruzioni:

- Nel pannello a sinistra andare sulla sezione **Tabelle** e successivamente su **Crea Tabella**
- Nella nuova pagina sarà possibile settare le impostazioni della nuova tabella, quindi inserire un nome e due chiavi primarie, una di partizione e una di ordinamento, come mostra l'immagine seguente

Crea una tabella di DynamoDB

DynamoDB è un database senza schema che richiede solo un nome di tabella e una chiave primaria. La chiave primaria della tabella è composta da uno o due attributi che identificano in modo univoco gli elementi, distribuiscono i dati e li ordinano in ciascuna partizione.

Nome tabella* CrispyEmployee

Chiave primaria* Chiave di partizione

Nome Stringa

Email Stringa

Impostazioni tabella

Le impostazioni predefinite offrono il modo più veloce per iniziare a utilizzare la tabella. Puoi modificare queste impostazioni definite ora o dopo la creazione della tabella.

Figura 3.7: Esempio creazione tabella AWS DynamoDB

- Infine cliccare su **Crea**

Ora è possibile popolare il database inserendo i dati nell'apposito pannello di controllo di DynamoDB oppure fare delle interrogazioni da codice.

3.3 Creazione della Skill

Una volta configurati correttamente tutti i servizi di AWS che Concierge Croccante andrà ad utilizzare si è proceduti con la realizzazione vera e propria della Skill. In questa parte infatti si andrà a mostrare e spiegare come questa sia stata realizzata basandosi sull'analisi della VUI fatta al punto 2.5. Il processo di realizzazione della Skill parte dalla console per sviluppatori di Alexa, a seguire la creazione della funzione Lambda dove verrà eseguito il programma, terminando infine con la stesura del codice mostrando parti di esso ritenute fondamentali.

3.3.1 Alexa Developer Console

Per creare la Skill C.C. è necessario visitare ed entrare nella console di gestione ed accedere al servizio Alexa Skills Kit⁹. Una volta fatto l'accesso con le proprie credenziali si è proceduto a creare il prodotto:

- Cliccare su **Create Skill** per creare la Skill;

Alexa Skills

The screenshot shows the Alexa Skills Kit dashboard. At the top, there is a search bar labeled "Search by skill name or skill ID" and a blue button labeled "Create Skill". Below the search bar is a table with the following columns: SKILL NAME, LANGUAGE, TYPE, MODIFIED, STATUS, and ACTIONS. There is one row in the table representing the skill "ConciergeCroccanteCrisp", which is listed as "Italian (IT)" in the LANGUAGE column, "Custom" in the TYPE column, and "2019-07-25" in the MODIFIED column. The STATUS column shows a green dot followed by "In Development". The ACTIONS column contains three links: "Analytics", "Edit", and "Delete".

- Nella pagina successiva, dare un nome alla propria Skill (facendo attenzione al fatto che quest'ultimo **non** sarà il nome di invocazione), scegliere la lingua, in questo caso *Italian (IT)*, e scegliere *Custom* come modello lasciando invariato il resto dei parametri. Successivamente cliccare nuovamente su **Create Skill**;

Skill name

Enter skill name

0/50 characters

Default language

Italian (IT)

More languages can be added to your skill after creation

Choose a model to add to your skill

There are many ways to start building a skill. You can design your own custom model or start with a pre-built model. Pre-built models are interaction models that contain a package of intents and utterances that you can add to your skill.

Custom SELECTED

Design a unique experience for your users. A custom model enables you to create all of your skill's interactions.

Flash Briefing

Give users control of their news feed. This pre-built model lets users control what updates they listen to.

"Alexa, quali sono le notizie del giorno?"

Smart Home

Give users control of their smart home devices. This pre-built model lets users turn off the lights and other devices without getting up.

"Alexa, accendi le luci in cucina"

⁹ Alexa Skill Kit. URL: <https://developer.amazon.com/it/alex-skills-kit>

- Si verrà reindirizzati nella pagina Alexa Developer Console dove sarà possibile customizzare la Skill. Nella sezione **Custom**, situata a sinistra, andare su **Invocation** per inserire il nome con cui si desidera invocare la Skill

The screenshot shows the Alexa Developer Console interface. On the left, there's a sidebar titled 'CUSTOM' with sections for 'Interaction Model' and 'Invocation'. Under 'Invocation', it says 'Intents (11)' and shows a list of intents categorized under 'GestionelIncontri': 'incontro', 'nomeDipendenteIncontri', 'nomeVisitatoreIncontri', and 'orario'. To the right, there's a main panel titled 'Invocation' with the sub-section 'Skill Invocation Name'. It contains a text input field with the value 'sono qui' and a note below it: 'User: Alexa, ask daily horoscopes for the horoscope for Gemini'. Below this, there's another text input field labeled 'Skill Invocation Name'.

- Successivamente è necessario creare gli intenti (*intents*), che rappresentano un'azione che soddisfa una richiesta del utente. Sempre nella sezione **Custom - JSON Editor** è possibile creare gli intenti caricando un file .json oppure scrivendoli sull'editor della pagina. A questo punto il comando di invocazione e gli intents sono stati impostati correttamente.

The screenshot shows the Alexa Developer Console interface with the 'JSON Editor' open. On the left, the 'CUSTOM' sidebar shows 'Invocation' and 'Intents (11)'. The 'JSON Editor' panel has a title 'JSON Editor' and a note 'Click here to learn more about the schema definition for interaction models.' Below this is a code editor area with a 'Drag and drop a .json file' placeholder. The JSON code in the editor is as follows:

```

1  {
2   "interactionModel": {
3     "languageModel": {
4       "invocationName": "sono qui",
5       "intents": [
6         {
7           "name": "AMAZON.CancelIntent",
8           "samples": [
9             "Cancella"
10            ]
11         },
12         {
13           "name": "AMAZON.HelpIntent",
14           "samples": [
15             "ho bisogno di aiuto",
16             "aiutami",
17             "cosa puoi fare"
18           ]
19         }
20       ]
21     }
22   }
23 }
```

3.3.1.1 Skill ID

Il passo seguente è stato quello di salvare lo Skill ID, la chiave identificativa della Skill appena creata, che nei passi successi servirà per impostare l'endpoint: ovvero collegare la Skill alla Lambda dove risiederà il codice. Per fare ciò è stato sufficiente identificare e salvare l'ID inglobato nell'indirizzo URL della Skill appena creata, come mostra l'immagine seguente:

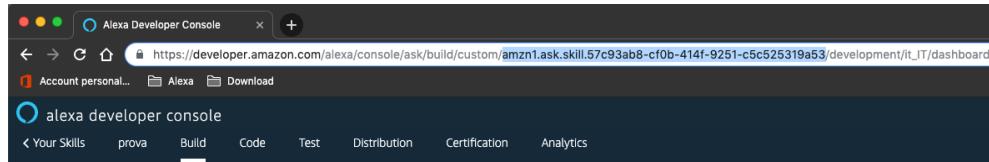


Figura 3.8: Esempio Skill ID

3.3.2 AWS Lambda

Amazon Lambda¹⁰ è un servizio della suite aws che consente di eseguire codice senza dover effettuare il provisioning¹¹ e gestire server. Con Amazon Lambda è quindi possibile eseguire codice per qualsiasi tipo di applicazione o servizio back-end, senza alcuna amministrazione. Caricato il codice, Lambda attua tutte le azioni necessarie per eseguirlo e ricalibrare le risorse con massima disponibilità. È possibile anche configurare il codice in modo che venga attivato automaticamente da altri servizi AWS oppure che venga richiamato direttamente da qualsiasi applicazione Web o mobile.

Una volta proceduti a creare la Skill la fase successiva è stata la realizzazione della funzione Lambda necessario per l'esecuzione del codice C.C.. Recandosi nell'omonima pagina, Amazon Lambda, sono stati eseguiti i passaggi di configurazione per creare la funzione ospitante il codice implementato o in fase di implementazione. Una volta fatto l'accesso con le proprie credenziali si è proceduto nel seguente modo:

¹⁰ AWS Lambda. URL: <https://aws.amazon.com/it/lambda/>

¹¹In telecomunicazioni è un servizi cloud a disposizione del utente che include hardware, software, cablaggio ed altro

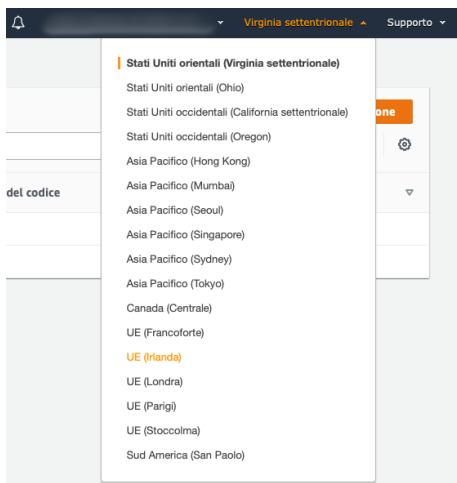


Figura 3.9: Selezione regione AWS Lambda

Prima di procedere è importante cambiare la regione di esecuzione della funzione Lambda. Quindi cliccare in alto a destra sulla regione (seconda voce) e scegliere *UE (Irlanda)*, come mostrato in figura. Questo passaggio è necessario perché in base alla regione scelta, AWS - Lambda mette a disposizione più o meno servizi da integrare con la funzione. Tali informazioni possono essere reperite nell'apposita pagina^a. Di seguito viene fornito anche URL delle **Regioni ed endpoint AWS**^b dove sono riportate le informazioni degli endpoint di ogni servizio per ciascuna regione.

^a<https://amzn.to/30GkmTd>

^b<https://amzn.to/2NCexm7>

- Cliccare in alto a destra su **Crea funzione** per creare la Lambda
- Nella pagina successiva, dare un nome alla funzione su cui verrà caricato il codice, nella sezione **runtime** scegliere *Node.js 10.x* e infine nella sezione **Autorizzazioni** - **Ruolo di esecuzione** scegliere *Utilizza ruolo esistente - ConciergeCroccanteRole* (ruolo creato in precedenza durante la configurazione di AWS IAM al punto [3.2.3](#)). L'immagine seguente mostra la scelta corretta delle impostazioni base.

- Successivamente cliccare nuovamente su **Crea funzione**
- Ora la funzione è stata impostata con le autorizzazioni sufficienti e necessarie per poter utilizzare i servizi che la Skill necessita. A destra cliccare su **Aggiungi trigger**, selezionare **Alexa Skill Kit**, inserire la Skill ID (salvata in precedenza) e cliccare su **Aggiungi**

3.3.2.1 Endpoint

Ora è possibile terminare il processo di configurazione della Skill eseguendo il collegamento tra Skill e Lambda. Ritornando nella console di AWS, sul servizio Lambda, è stato copiato l'indirizzo ARN della funzione, situato in alto a destra, come mostra nell'immagine seguente:

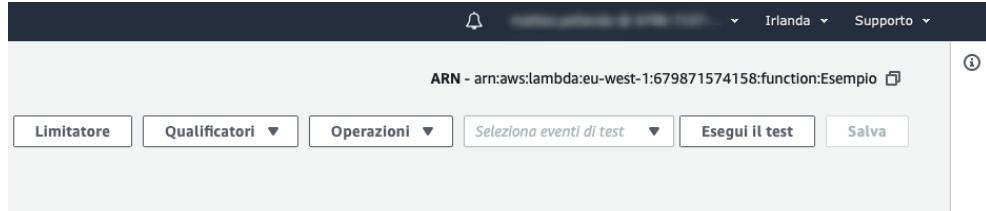


Figura 3.10: Esempio URL ARN Lambda

Successivamente, nella Skill in Alexa Developer Console, nella sezione **Custom - Endpoint**, incollare l'URL ARN della Lambda copiato in **Default Region**.



Figura 3.11: Esempio URL ARN Lambda

Infine cliccare su **Save Endpoint**. Ora la Skill l'endpoint impostato con l'esecuzione della funzione lambda creata.

Non rimane altro che caricare il codice del prodotto. Per farlo basterà recarsi nuovamente sulla console di AWS nel servizio Lambda, scegliere la propria funzione e su **Codice della funzione** scegliere *Carica un file .zip* per caricare la cartella compressa del progetto.

The screenshot shows the AWS Lambda function code editor. At the top, it says 'Codice della funzione' and 'Informazioni'. Below that, 'Tipo di voce del codice' is set to 'Modifica codice inserito' and 'Runtime' is set to 'Node.js 10.x'. The code editor displays the 'index.js' file with the following content:

```

1 exports.handler = async (event) => {
2     // TODO implement
3     const response = {
4         statusCode: 200,
5         body: JSON.stringify('Hello from Lambda!'),
6     };
7     return response;
8 }
9

```

Figura 3.12: Codice funzione

3.4 Organizzazione del codice

Successiva alla parte di comprensione e configurazione dei servizi utilizzati in questo progetto, e alla creazione della Skill, è stato eseguito un micro-processo di organizzazione del codice prima della sua stesura. Questa pratica adottata è stata necessaria per avere uno schema e una gerarchia delle parti di codice che sono state sviluppate durante il periodo di tirocinio. Tale organizzazione ha permesso di sviluppare in maniera più ordinata e consapevole riducendo la probabilità di introdurre errori e rendendo il codice più manutenibile. Pertanto la cartella del progetto è stata così disposta:

- **Skill:** cartella principale del progetto (main folder) contenente al suo interno altre cartelle e file organizzati secondo una determinata gerarchia
 - > **apl-data:** cartella contenente i dati delle schermate APL
 - > **apl-template:** cartella contenente i template lo stile delle schermate APL
 - > **handlers:** cartella contenente i file .js di ogni handlers (gestori) creati nella Skill
 - > **i18n:** cartella contenente file .js con le traduzioni della lingua
 - > **node_modules:** cartella contenente tutti i pacchetti installati con il gestore npm
 - > **task:** cartella contenente file .js da eseguire a riga di comando per l'esecuzione di funzionalità primitive
 - > **utils:** cartella contenente file .js con classi di utilità, come lettura del db, invio di e-mail, ecc..
 - **credentials.json:** file contenente delle credenziali per accedere al servizio di Google Calendar
 - **googleAuthTokenCredential.json:** file contenente delle credenziali per accedere al servizio di Google Calendar
 - **index.js:** file principale dove inizia l'esecuzione della Skill. La Lambda difatti partirà con la lettura di questo file
 - **intent.json:** file .json contenente tutti gli intent creati per la Skill
 - **package.-lock.json:** file di configurazione automaticamente creato dal gestore di pacchetti npm
 - **package.json:** file di configurazione automaticamente creato dal gestore di pacchetti npm
 - **policy.json:** file .json contenente le regole di policy create al punto [3.2.3](#)

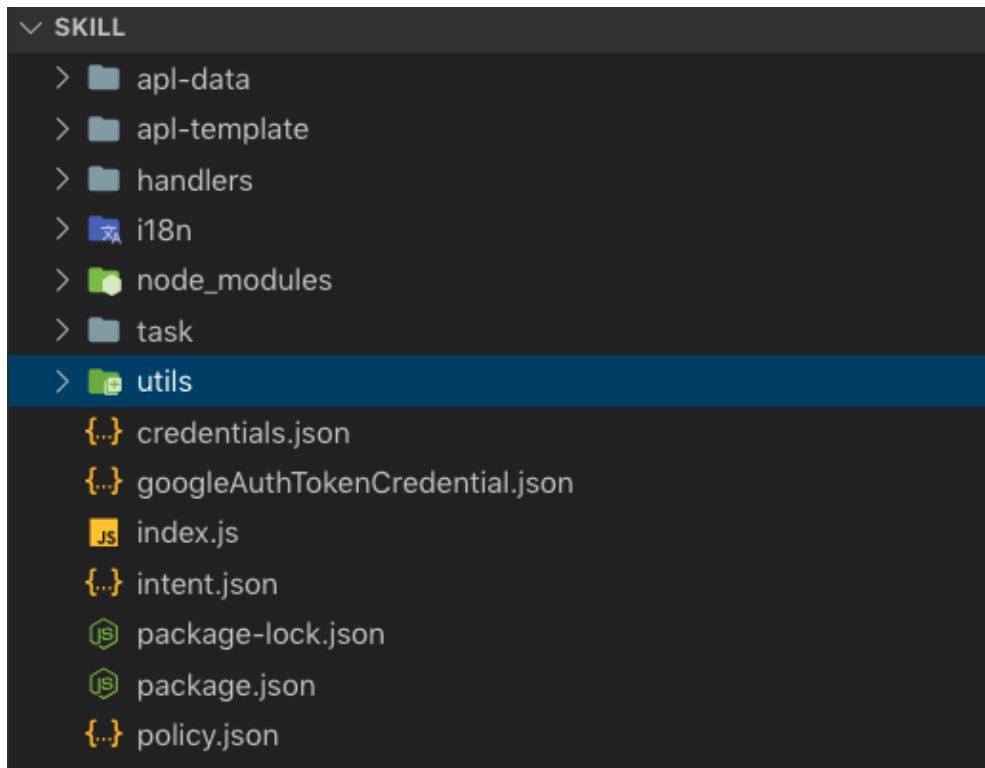


Figura 3.13: Gerarchia cartella progetto C.C.

3.4.1 Handlers

Come descritto prima la cartella del progetto C.C. è stata disposta per una gestione ordinata del codice. Una delle sotto-cartelle contenute al suo interno è dedicata per la gestione degli handelrs. Nel codice della Skill gli handerls sono delle funzioni che hanno lo scopo di gestire gli intenti che vengono scatenati all’invocazione di un comando da parte dell’utente. Gli handlers sono stati così realizzati:

- **index.js**: che contiene ed ingloba l’export di tutte le classi dei handlers contenuti in questa cartella
- **gestioneConsegne_Complete.js**: questo handler ha il compito di catturare l’evento nel caso in cui l’utente intenda consegnare un pacco (o lettera) sapendo già il nome del destinatario e, o l’oggetto consegnato o la professione dell’utente che sta conversando
- **gestioneConsegne_NomeDipendete.js**: questo handler ha il compito di catturare l’evento nel caso in cui l’utente intenda consegnare un pacco (o lettera) senza sapere già il nome del destinatario. Questo gestore ha il compito di richiedere tale dato domandandolo all’utente che sta conversando con la Skill
- **gestioneIncontri_Complete.js**: questo handler ha il compito di catturare l’evento nel caso in cui l’utente visitatore abbia un appuntamento e i dati necessari sono stati tutti raccolti

- **gestioneIncontri_NomeDipendente_NomeVisitatore.js**: questo handelr ha il compito di catturare l'evento nel caso in cui l'utente visitatore abbia un appuntamento e, il nome del dipendente cercato e del cliente appena entrato, sono stati raccolti. Il gestore eseguirà dei controlli per verificare l'attendibilità dei dati per poi proseguire con il dialogo oppure chiedendone di nuovi dati
- **gestioneIncontri_NomeDipendente.js**: questo handler ha il compito di catturare l'evento nel caso in cui l'utente abbia un appuntamento e la Skill richiede il nome del dipendente cercato
- **gestioneIncontri_Orari.js**: questo handler ha il compito di catturare l'evento nel caso in cui l'utente abbia un appuntamento e la Skill richiede l'orario per verificare l'evento nel calendario
- **vorreiIncontri_Complete.js**: questo handler ha il compito di catturare l'evento nel caso in cui l'utente visitatore desideri incontrare un dipendente e sono stati raccolti tutti i dati necessari
- **vorreiIncontri_CompleteWithOrario.js**: questo handler ha il compito di catturare l'evento nel caso in cui l'utente visitatore desideri incontrare un dipendente e richiede l'orario di appuntamento per verificare l'esistenza dell'evento nel calendario
- **vorreiIncontri_NomeDipendente_NomeVisitatore.js**: questo handler ha il compito di catturare l'evento nel caso in cui l'utente visitatore desideri incontrare un dipendente e, il nome del dipendente cercato e del cliente appena entrato, sono stati raccolti. Il gestore eseguirà dei controlli per verificare l'attendibilità dei dati per poi proseguire con il dialogo oppure chiedendone di nuovi dati
- **vorreiIncontri_NomeDipendente.js**: questo handler ha il compito di catturare l'evento nel caso in cui l'utente visitatore desideri incontrare un dipendente e la Skill richiede il nome di quest'ultimo
- **vorreiIncontri_SiNo.js**: questo handler ha il compito di catturare l'evento nel caso in cui l'utente visitatore abbia un appuntamento e la Skill ne richiede l'orario per verificare l'evento nel calendario
- **touchWrapperUserEvent.js**: questo handler ha il compito di catturare gli eventi scatenati dal tocco sul display

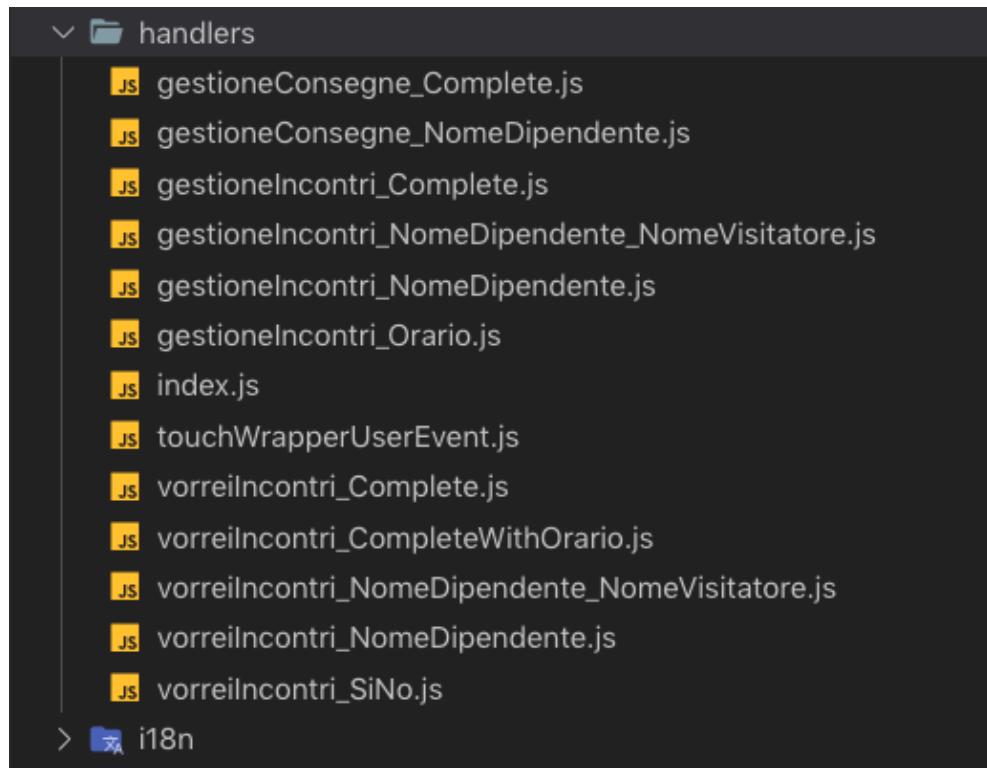


Figura 3.14: Gerarchia cartella progetto C.C. - Handlers

3.4.2 Utils

Lo stesso criterio usato per gli handlers è stato adottato anche per le classi Javascript che contengono il codice delle funzionalità della Skill. La cartella *utils* contiene appunto quei metodi che permettono ad esempio la lettura del database, l'invio di e-mail, l'invio di notifiche e la lettura del calendario. Le utilità sono state così realizzate:

- **databaseUtils.js**: questa classe mette a disposizione i metodi per interrogare il database e sono presenti le seguenti funzioni:

```
> getInstance()
> runQuery(params: var)
> runScan(params: var)
> runPut(params: var)
> checkEmployee(nome_dipendente: var)
```

- **emailUtils.js**: questa classe mette a disposizione i metodi per l'invio di messaggi e-mail ed è presente la seguente funzione:

```
> sendMail(from: var, to: var, message: var, oggetto: var)
```

- **gcalendarUtils.js**: questa classe mette a disposizione i metodi per la lettura e verifica di eventi nel calendario e sono presenti le seguenti funzioni:

```
> listEvents(idCalendar: var)
> checkEvent(events: var, email_dipendente: var, nome_visitatore: var, orarioEvento: var)
> generateOAuth2Client()
```

- **gchatUtils.js**: questa classe mette a disposizione i metodi per l'invio di notifiche su Google Chat, funzionalità in fase di sviluppo fatta nell'ultimo periodo di tirocinio, e sono presenti le seguenti funzioni:

```
> sendNotify(type: var, message: var)
> checkThread(type: var)
> putThread(thread: var, thread_type: var)
```

- **prepareQuery.js**: questa classe mette a disposizioni i metodi per ricevere le query di interrogazione che si desidera e sono presenti le seguenti funzioni:

```
> scanEmployee()
> queryEmployee(nomeDipendente: var)
> putThread()
> updateThread(old_thread: var, new_thread: var)
```

- **slacknotifyUtils.js**: questa classe mette a disposizione i metodi per l'invio di notifiche su Slack e al suo interno sono presenti le seguenti funzioni:

```
> sendNotify(channel: var, message: var)
> sendDirectlyNotify(member: var, MESSAGE: var)
> addTag(toTag: var)
> listMembersApl(intent: var)
> getIdMember(toTag: var)
> membersList()
```

- **abracadabraUtils.js**: questo metodo contiene un easter egg¹²

¹²Un easter egg in informatica è un contenuto di natura bizzarra e innocuo che gli sviluppatori nascondono all'interno del prodotto

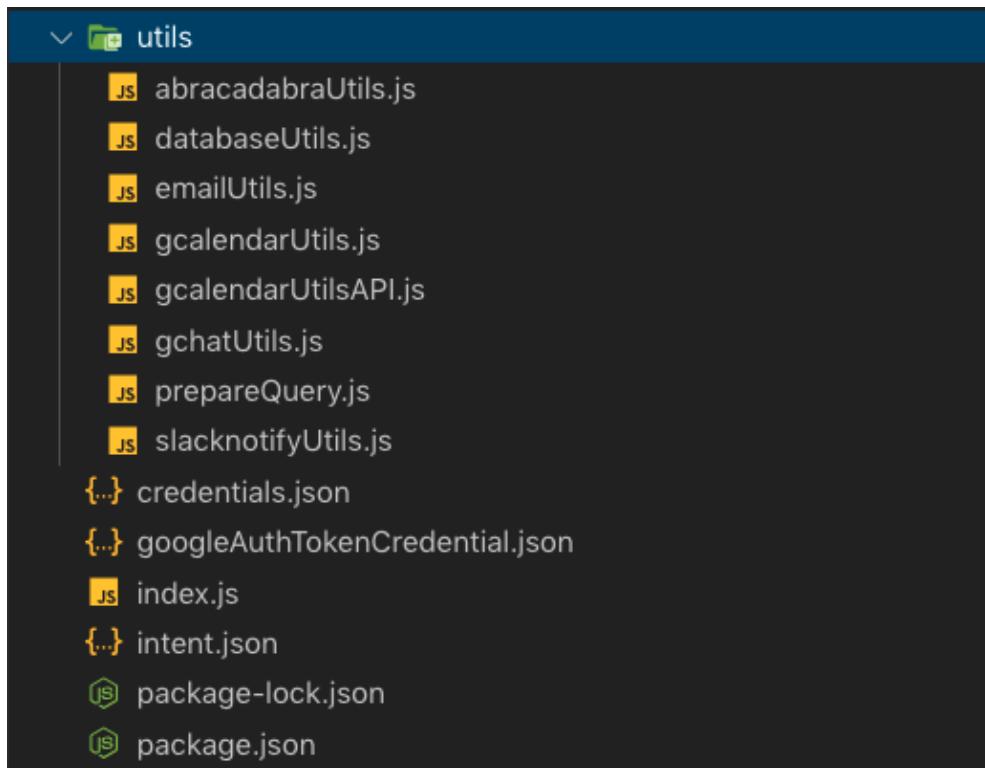


Figura 3.15: Gerarchia cartella progetto C.C. - Utils

3.5 Pacchetti

Particolare attenzione va fatta sulla gestione dei pacchetti e quali di essi sono stati installati nel progetto Concierge Croccante. A tale necessità è stato utilizzato il gestore di pacchetti npm, menzionato in precedente al punto 1.5.4, che ha consentito di installare in maniera sicura e immediata i pacchetti necessari. Installando tali pacchetti il gestore ha suddiviso il codice all'interno di una directory chiamata `node_modules` creando inoltre un file di configurazione delle installazioni fatte.

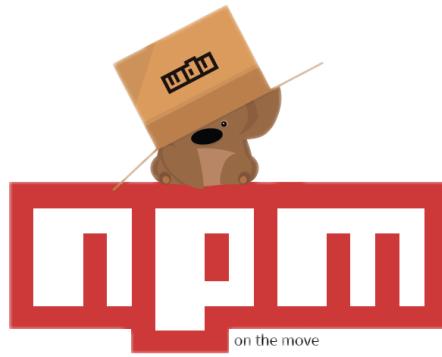


Figura 3.16: Logo NPM

In questa fase di progetto si è provveduto ad installare npm e tutti i pacchetti necessari alla Skill Concierge Croccante nel seguente modo:

- Dalla documentazione di npm¹³ viene riportato il seguente comando da eseguire su terminale:

```
1 $ [sudo] npm install npm -g
```

Il comando installerà l'ultima versione del gestore pacchetti. Se si desidera verificare e visualizzare l'ultima versione di npm è necessario eseguire il comando:

```
1 $ npm -v
```

- I pacchetti installati nel progetto sono stati i seguenti:

> **ask-sdk** e **ask-sdk-core** di AWS che semplifica la creazione di Skill, permettendo di utilizzare le funzionalità base di Alexa in maniera rapida:

```
1 $ npm install --save ask-sdk
2 $ npm install --save ask-sdk-core
```

> **request** per semplificare ed effettuare chiamate http, supporta HTTPS i re-indirizzamenti:

```
1 $ npm install request
```

> **googleapis** è una libreria client per l'utilizzo delle API di Google, rendendo più semplice ed immediate le chiamate ai servizi come Calendar:

```
1 $ npm install googleapis
```

> **slack-notify** è una libreria che rende flessibile l'utilizzo dell'API Webhooks¹⁴ di Slack, semplificando l'invio di notifiche a Slack dalla Skill:

```
1 $ npm install slack-notify
```

> **i18next** e **i18next-sprintf-postprocessor** sono dei framework di internazionalizzazione popolari ambienti javascript:

```
1 $ npm install i18next
2 $ npm install i18next-sprintf-postprocessor
```

¹³Doc npm. URL: <https://docs.npmjs.com/downloading-and-installing-node-js-and-npm>

¹⁴Incoming Webhooks. URL: <https://api.slack.com/incoming-webhooks>

3.6 Calendario

Per la soddisfazione del requisito RO17 analizzato al punto [2.3](#), ovvero la lettura e verifica degli appuntamenti in calendario, è stato richiesto l'uso di Google Calendar, già utilizzato dall'azienda Crispy Bacon per le sue note funzionalità.

3.6.1 Google Calendar

Per l'integrazione col servizio di Google Calendar sono a disposizione delle API documentate reperibili al indirizzo developers.google.com/calendar/. Dalla guida emergono i seguenti requisti necessari:



Figura 3.17: Icona Google Calendar

- Node.js
- googleapis package
- Account Google funzionante e verificato
- Autorizzazione OAuth^a

^aOAuth: protocollo di rete open e standard, progettato per lavorare con il protocollo HTTP.

Nei punti successivi verranno analizzati aspetti importanti, con esempi di codice, riguardante l'integrazione con il calendario in questione e l'uso delle sue API.

3.6.1.1 OAuth2

Per consentire la lettura ed ottenere la lista di eventi presenti nel calendario è stato necessario eseguire un'autenticazione tramite il protocollo OAuth 2.0¹⁵. Quest'ultimo è un protocollo di rete open che consente l'emissione di un token di accesso da parte di un server autorizzato ad un client di terze parti, nel nostro caso il servizio di Google Calendar, previa approvazione dell'utente proprietario della risorsa cui si intende accedere. Per poter ottenere ciò sono stati svolti i seguenti passi:

- Visitando la pagina Google Calendar API¹⁶ è stato abilitato l'utilizzo delle API per il calendario. Nella pagina infatti è presente il primo step da seguire dal titolo *"Turn on the Google Calendar API"* dove presente il bottone adibito all'abilitazione:

Step 1: Turn on the Google Calendar API



Click this button to create a new Cloud Platform project and automatically enable the Google Calendar API:

[ENABLE THE GOOGLE CALENDAR API](#)

In resulting dialog click **DOWNLOAD CLIENT CONFIGURATION** and save the file `credentials.json` to your working directory.

¹⁵OAuth 2.0. URL: <https://oauth.net/2/>

¹⁶G. Calendar API. URL: <https://developers.google.com/calendar/quickstart/nodejs>

Nella sezione corrente, una volta cliccato il bottone **ENABLE THE GOOGLE CALENDAR API** ed eseguito l'accesso con l'account Google il quale si desidera leggere gli eventi in calendario, è stato salvato il file `credentials.json` ottenuto dall'accesso e fornito nella cartella di progetto.

- Sempre nella stessa pagina è stato copiato ed eseguito l'esempio di set up riportato per ottenere l'OAuth necessario a visualizzare gli eventi nel calendario. Di seguito si riporta tale esempio di codice, importante appunto per ricevere le autorizzazioni necessarie:

```

1 const fs = require('fs');
2 const readline = require('readline');
3 const {google} = require('googleapis');
4 const path = require('path');
5
6 const SCOPES = ['https://www.googleapis.com/auth/calendar.readonly'];
7 const TOKEN_PATH = path.join(__dirname, '../googleAuthTokenCredential
     .json');
8
9 // Load client secrets from a local file.
10 fs.readFile(path.join(__dirname, '../credentials.json'), (err,
    content) => {
11   if (err) return console.log('Error loading client secret file:',
     err);
12   // Authorize a client with credentials, then call the Google
     Calendar API.
13   authorize(JSON.parse(content), listEvents);
14 });
15
16 /**
17 * Create an OAuth2 client with the given credentials, and then
     execute the
18 * given callback function.
19 * @param {Object} credentials The authorization client credentials.
20 * @param {function} callback The callback to call with the
     authorized client.
21 */
22 function authorize(credentials, callback) {
23   const {client_secret, client_id, redirect_uris} = credentials.
     installed;
24   const oAuth2Client = new google.auth.OAuth2(
25     client_id, client_secret, redirect_uris[0]);
26
27   // Check if we have previously stored a token.
28   fs.readFile(TOKEN_PATH, (err, token) => {
29     if (err) return getAccessToken(oAuth2Client, callback);
30     oAuth2Client.setCredentials(JSON.parse(token));
31     callback(oAuth2Client);
32   });
33 }
34
35 /**
36 * Get and store new token after prompting for user authorization,
     and then
37 * execute the given callback with the authorized OAuth2 client.
38 * @param {google.auth.OAuth2} oAuth2Client The OAuth2 client to get
     token for.
39 * @param {getEventsCallback} callback The callback for the
     authorized client.
40 */
41 function getAccessToken(oAuth2Client, callback) {

```

```

42 const authUrl = oAuth2Client.generateAuthUrl({
43   access_type: 'offline',
44   scope: SCOPES,
45 });
46 console.log('Authorize this app by visiting this url:', authUrl);
47 const rl = readline.createInterface({
48   input: process.stdin,
49   output: process.stdout,
50 });
51 rl.question('Enter the code from that page here: ', (code) => {
52   rl.close();
53   oAuth2Client.getToken(code, (err, token) => {
54     if (err) return console.error('Error retrieving access token', err);
55     oAuth2Client.setCredentials(token);
56     // Store the token to disk for later program executions
57     fs.writeFile(TOKEN_PATH, JSON.stringify(token), (err) => {
58       if (err) return console.error(err);
59       console.log('Token stored to', TOKEN_PATH);
60     });
61     callback(oAuth2Client);
62   });
63 });
64 }
65 /**
66 * Lists the next 10 events on the user's primary calendar.
67 * @param {google.auth.OAuth2} auth An authorized OAuth2 client.
68 */
69 function listEvents(auth) {
70   const calendar = google.calendar({version: 'v3', auth});
71   calendar.events.list({
72     calendarId: 'primary',
73     timeMin: (new Date()).toISOString(),
74     maxResults: 10,
75     singleEvents: true,
76     orderBy: 'startTime',
77   }, (err, res) => {
78     if (err) return console.log('The API returned an error: ' + err);
79     ;
80     const events = res.data.items;
81     if (events.length) {
82       console.log('Upcoming 10 events:');
83       events.map((event, i) => {
84         const start = event.start.dateTime || event.start.date;
85         console.log(`${start} - ${event.summary}`);
86       });
87     } else {
88       console.log('No upcoming events found.');
89     }
90   });
}

```

Listing 3.2: Esempio Set up Google Calendar

- Per eseguire tale esempio di codice è stato sufficiente aprire il terminale sulla cartella di progetto dove risiede il file di set up e lanciare lo script con il seguente comando:

```
1 $ node [filename]
```

Nel caso in cui fossa la prima volta che eseguito tale esempio è necessario autorizzare l'accesso:

1. Aprire l'URL restituito dal terminale col proprio web browser
2. Fare il login in se non si è già loggati
3. Cliccare il bottone **Accept**
4. Copiare il codice ricevuto, incollarlo nella terminale e premere **Enter**

3.6.1.2 Lettura eventi

All'interno del progetto Concierge Croccante la fase di lettura del calendario è stata semplificata realizzando una classe contenente al suo interno metodi appositi. Di tali metodi, già menzionati al punto 3.4.2, ne viene riportato il codice e analizzato il loro funzionamento:

```

1 const {google} = require('googleapis');
2 const googleCredentialToken = require('../googleAuthTokenCredential.json'
  );
3 const googleCredentials = require('../credentials.json');
4
5 class gcalendarUtils{
6   static async listEvents(idCalendar) {
7     let now = new Date(); now.setHours(1);
8     const iso_timeMin = now.toISOString();
9     now.setHours(23);
10    const iso_timeMax = now.toISOString();
11
12    const auth = generateOAuth2Client();
13    const calendar = google.calendar({
14      version: 'v3',
15      auth: auth
16    });
17    const events = await calendar.events.list({
18      calendarId: idCalendar,
19      timeMin: iso_timeMin,
20      timeMax: iso_timeMax,
21      maxResults: 15,
22      singleEvents: true,
23      orderBy: 'startTime',
24    });
25    return events.data.items;
26  }
27
28  /* Pre: checkEvent riceve in ingresso un elenco di eventi,
29   *       il nome della visitatore e l'orario del evento. */
30  static checkEvent(events, email_dipendente, nome_visitatore,
31    orarioEvento){ }
32  /* Post: ritornerà false se nessun vento è presente in calendario,
33         ritornerà 1 se il visitatore indica l'orario corretto,
34         ritornerà 0 se il visitatore indica un orario errato ma
35         comunque l'evento a lui associato esiste. */
36}
37 function generateOAuth2Client () {
38   const { client_secret, client_id, redirect_uris } =
39     googleCredentials.installed;
40   const oAuth2Client = new google.auth.OAuth2(client_id, client_secret
41     , redirect_uris[0]);
42   oAuth2Client.setCredentials(googleCredentialToken);
43   return oAuth2Client;
44 };
45 module.exports = gcalendarUtils;

```

Listing 3.3: Metodi listEvent e generateOAuth2Client

Per ottenere la lista di eventi non servirà altro che richiamare `listEvents(idCalendar)`. La funzione ritornerà un oggetto contenente tutti gli eventi, con le loro informazioni, della giornata odierna fino ad un massimo di 15 risultato. Attenzione: i file citati `credentials.json` e `googleAuthAccessTokenCredential.json` non sono altro che le credenziali recuperate nei punti precedenti per accedere con il token OAuth.

Altro metodo presente e largamente utilizzato è `checkEvent(events, email_dipendente, nome_visitatore, orarioEvento)`:

```

1 const {google} = require('googleapis');
2 const googleCredentialToken = require('../googleAuthAccessTokenCredential.json
3   ');
4 const googleCredentials = require('../credentials.json');
5
6 class gcalendarUtils{
7   static async listEvents(idCalendar) { }
8
9   /* Pre: checkEvent riceve in ingresso un elenco di eventi,
10      il nome della visitatore e l'orario del evento. */
11  static checkEvent(events, email_dipendente, nome_visitatore,
12    orarioEvento){
13    if (events.length){
14      const array_nome_vis = nome_visitatore.split(' ');
15
16      /* Scorro l'oggetto contente gli eventi come se fosse un map.
17       * Altrimenti l'oggetto event viene considerato vuoto. */
18      const out_event = events.find(event => {
19        let bools_check_nome_cognome = [false, false];
20        let bool_check_dip = false;
21        let array_event_summary = event.summary.split(',');
22
23        /* Scorro l'arry dal i=0 a i<n per verificare la
24         presenza
25         * del nome e cognome del visitatore. */
26        for(let i=0; i<array_event_summary.length && (
27          bools_check_nome_cognome[0] === false || 
28          bools_check_nome_cognome[1] === false); ++i){
29          if(array_event_summary[i].toLowerCase() ===
30            array_nome_vis[0].toLowerCase())
31            bools_check_nome_cognome[0] = true;
32
33          if(array_event_summary[i].toLowerCase() ===
34            array_nome_vis[1].toLowerCase())
35            bools_check_nome_cognome[1] = true;
36        }
37
38        if(event.attendees){
39          const out = event.attendees.find(attendee => {
40            return attendee.email === email_dipendente;})
41          if(out)
42            bool_check_dip = true;
43        }
44
45        return bools_check_nome_cognome[0] &&
46          bools_check_nome_cognome[1] && bool_check_dip;
47      });
48
49      // Se non ho nessuna corrispondenza nome cognome orario
50      if(!out_event)
51        return false;
52      // Se ho corrispondenza nome cognome orario

```

```

44         const startEvent = out_event.start.dateTime.slice(11,
45             16);
46         if(startEvent === orarioEvento){
47             console.log('[gcalendarUtils.checkEvent]: ${
48                 startEvent} - ${out_event.summary}');
49             return true;
50         }
51         // Se ho corrispondenza nome cognome
52         else
53             return startEvent;
54     }
55     else
56         return false;
57 }
58 /* Post: ritornerà false se nessun vento è' presente in calendario,
59    ritornerà 1 se il visitatore indica l'orario corretto,
60    ritornerà 0 se il visitatore indica un orario errato ma
61    comunque l'evento a lui associato esiste. */
62
63 function generateOAuth2Client () {};
64 module.exports = gcalendarUtils;

```

Listing 3.4: Metodo checkEvent

Questo metodo pone come precondizione il fatto di ricevere in ingresso l'elenco di eventi ed i valori: nome visitatore, nome persona cercata ed orario; non vuoti. Il risultato sarà un valore *boolean false* nel caso **non** sia presente alcun evento nel calendario, un valore numerico *1* nel caso sia stato trovato un evento per la persona e l'orario indicato, oppure *0* se esiste un evento ma l'orario indicato non coincide con quello effettivo.

3.7 Invio notifiche

Altra specifica per la soddisfazione dei requisiti RO18 e RO19 analizzati al punto 2.3, è l'invio di notifiche. Per ovviare a ciò sono state realizzate le funzionalità per l'invio di messaggi istantanei via Slack e e-mail via AWS SES visto il già utilizzo di questi servizi da parte di Crispy Bacon.

3.7.1 Slack

Per l'integrazione col servizio di Slack sono a disposizione delle API documentate reperibili al indirizzo api.slack.com/. Dalla guida emergono i seguenti requisti necessari:



- Node.js
- slack-notify package
- Account Slack
- Incoming Webhook

Figura 3.18: Icona servizio - Slack

Nei punti successivi verranno analizzati aspetti importanti, con esempi di codice, riguardante l'invio di notifiche con Slack e l'uso delle sue API.

3.7.1.1 Incoming Webhooks

Per consentire l'invio di messaggi di notifica via chat viene scelto di utilizzare un bot con Incoming Webhook¹⁷. Quest'ultimo è un mezzo con il quale si semplifica l'invio di messaggi da un'applicazione a Slack. La creazione di un Incoming Webhook fornisce un URL univoco a cui si invia un payload JSON con il testo del messaggio e alcune opzioni. Per ottenere l'Incoming Webhook sono stati eseguiti i seguenti passaggi:

- Visitando la pagina dedicata api.slack.com/incoming-webhooks è stata creata un'App per Slack cliccando sul bottone "Create your Slack app":

1. Create a **Slack app** (if you don't have one already)

You won't get very far without doing this step, but luckily it's very simple, we even have a nice green button for you to click for it:

Create your Slack app

Si viene successivamente indirizzati in una nuova pagina dove sarà possibile creare la propria App per Slack:

1. Cliccare sul bottone **Create New App**
2. Inserire il nome dell'app
3. Scegliere lo *space* di lavoro dove installare l'app
4. Cliccare su **Crea App**

¹⁷<https://it.wikipedia.org/wiki/Webhook>

- Dopo aver creato l’App è stato abilitato l’Incoming Webhook nell’apposita pagina delle impostazioni. Fatto sono state visualizzate alcune opzioni aggiuntive. Tra queste era presente il pulsante *Aggiungi nuovo Webhook a Workspace*. Viene quindi aggiunto il Webhook ad canale/space di Slack designato per pubblicare l’App e salvata tale modifica facendo clic su **Autorizza la tua app**.

3.7.1.2 Invio notifiche

All’interno del progetto Concierge Croccante la fase di notifica via Slack è stata semplificata realizzando una classe contenente al suo interno metodi appropriati. Di tali metodi menzionati al punto 3.4.2 ne viene riportato il codice e analizzato il loro funzionamento¹⁸:

```

1 const MY_SLACK_WEBHOOK_URL = 'https://hooks.slack.com/services/
  xxxxxxxxxxxxxxxx/xxxxxxxxxxxx';
2 const TOKEN = "oxxb-117197354337-xxxxxxxxxxxx-120EClhwYXKKaj5ulGEo3vCh"
  ;
3 const request = require('request');
4 const slack = require('slack-notify')(MY_SLACK_WEBHOOK_URL);
5
6 class slacknotifyUtils {
7   static sendNotify(channel, message) {
8     slack.send({
9       link_names: 1,
10      channel: channel,
11      text: message
12    }, function (err) {
13      if (err) {
14        console.log("[slacknotifyUtils.sendNotify]: API error:",
15                    err);
16        throw err;
17      } else {
18        console.log("[slacknotifyUtils.sendNotify]: Slack
          notification sent!");
19      }
20    });
21
22   static async sendDirectlyNotify(member, MESSAGE){ }
23   static async addTag(toTag) { }
24   static async listMembersApl(intent) { }
25 }
26 async function getIdMember(toTag) { }
27 function membersList() { }
28 module.exports = slacknotifyUtils;
```

Listing 3.5: Metodo sendNotify

Il codice sopra riportato mostra il metodo `sendNotify(channel: var, message: var)`: tale funzione non fa altro che inviare il testo del messaggio nel canale di Slack desiderato passando i dati per parametro. La variabile costante `slack.send` richiama il metodo fornito del pacchetto `slack-notify` che non farà altro che fare una *request* in *POST* all’API `chat.postMessage`. Nel caso in cui il metodo riscontrasse un errore nell’invio del messaggio viene lanciata un’eccezione.

¹⁸NB: la variabile `MY SLACK WEBHOOK URL` contiene l’URL del Webhook dell’App

```

1 const MY_SLACK_WEBHOOK_URL = 'https://hooks.slack.com/services/
2   xxxxxxxxxxxxxxxxx/xxxxxxxxxxxxx';
3 const TOKEN = "xoxb-117197354337-xxxxxxxxxxxx-120EClhwYXKKaj5ulGEo3vCh"
4 ;
5 const request = require('request');
6 const slack = require('slack-notify')(MY_SLACK_WEBHOOK_URL);
7
8 class slacknotifyUtils {
9   static sendNotify(channel, message) { }
10
11   static async sendDirectlyNotify(member, MESSAGE){
12     let CHANNEL;
13     if(member.includes('@'))
14       CHANNEL = member.replace('<@', '').replace('>', '');
15     else
16       CHANNEL = await getIdMember(member);
17
18     return new Promise((resolve, reject) =>{
19       request.post("https://slack.com/api/chat.postMessage?token="
20         + TOKEN + "&channel=" + CHANNEL + "&text=" + MESSAGE + "
21         &pretty=1", function(error, response, body){
22           if(error){
23             console.log("[slacknotifyUtils.sendDirectlyNotify]:"
24               "$error", error);
25             reject(error);
26           }
27           else
28             resolve(body);
29         });
30     })
31
32   static async addTag(toTag) { }
33   static async listMembersApl(intent) { }
34 }
35
36 async function getIdMember(toTag) { }
37 function membersList() { }
38 module.exports = slacknotifyUtils;

```

Listing 3.6: Metodo sendDirectlyNotify

Per l'invio delle notifiche in direct message¹⁹ è stato messo a disposizione il metodo `sendDirectlyNotify(member: var, MESSAGE: var)`: tale funzione invierà il testo del messaggio al destinatario indicato alla funzione facendo, anche in questo caso, una *request* in *POST* all'API `chat.postMessage`. Differentemente dal metodo precedente qui non viene utilizzata alcuna libreria, ma viene usata l'API in maniera diretta.

¹⁹Messaggio privato inviato in un social media

```

1 const MY_SLACK_WEBHOOK_URL = 'https://hooks.slack.com/services/
  xxxxxxxxxxxxxxxxx/xxxxxxxxxxxxxx';
2 const TOKEN = "xoxb-117197354337-xxxxxxxxxxxx-120EClhwYXKKaj5ulGEo3vCh"
  ;
3 const request = require('request');
4 const slack = require('slack-notify')(MY_SLACK_WEBHOOK_URL);
5
6 class slacknotifyUtils {
7   static sendNotify(channel, message) { }
8   static async sendDirectlyNotify(member, MESSAGE){ }
9
10  static async addTag(toTag) {
11    let tagged = await getIdMember(toTag);
12
13    if(tagged === false)
14      tagged = toTag;
15    else
16      tagged = "<@" + tagged + ">";
17
18    return tagged;
19  }
20
21  static async listMembersApI(intent) { }
22}
23async function getIdMember(toTag) { }
24function membersList() { }
25module.exports = slacknotifyUtils;

```

Listing 3.7: Metodo addTag

Altra funzionalità messa a disposizione dalla classe è il metodo `addTag(toTag: var)`: tale funzione non fa altro che richiamare una funzione ausiliaria e privata della classe, più elaborata e non visibile all'utente esterno, che va ad aggiungere, grazie all'id del membro cercato, il tag necessario per menzionarlo all'interno delle chat di gruppo.


```

58         "item": {
59             "type": "Text",
60             "text": "- " + "Persona non trovata".toUpperCase(),
61             "fontSize": 25,
62             "color": "black"
63         },
64         "onPress": [
65             {
66                 "type": "SendEvent",
67                 "arguments": [
68                     "not found",
69                     "matteo.pellanda@crispybacon.it",
70                     intent,
71                     "matteo.pellanda@crispybacon.it"
72                 ]
73             }
74         ]
75     /* Concatenzazione del APL apl-touchwrapper-list-members con
76        l'array creato sopra */
77     let apl_touchwrapper_list = require("../apl-template/apl-
78         touchwrapper-list-members.json");
79     if (!apl_touchwrapper_list) {
80         console.log("[slacknotifyUtils.listMembersApl]: impossible to require, missing file apl-touchwrapper-
81             list-members.json");
82         return require('../apl-template/apl-touchwrapper-error')
83         ;
84     } else {
85         apl_touchwrapper_list.mainTemplate.items[1].items[2] .
86         items[0].items = array_employee_apl_obj;
87     }
88 }
89 async function getIdMember(toTag) { }
90 function membersList() { }
91 module.exports = slacknotifyUtils;

```

Listing 3.8: Metodo listMembersApl

Metodo più rilevante è `listMembersApl(intent: var)`: tale metodo viene richiamato dalla Skill qualora è necessaria la costruzione di un APL con elementi touch che al momento del tocco scatenino un evento. In questo caso l'APL che si va a creare è una lista dei membri di Crispy Bacon dove è possibile selezionare il nome per fornirlo alla Skill.

3.7.2 E-mail

Per l'invio di notifiche via e-mail, come già detto in precedenza, è stato utilizzato il servizio Amazon SES e la sua integrazione con il servizio sono necessari i seguenti requisiti:



**Simple
Email
Service**

- Node.js
- npm aws-sdk
- Account AWS
- Indirizzo e-mail

Figura 3.19: Icona servizio - SES

Nei punti successivi verranno analizzati aspetti importanti, con esempi di codice, riguardante l'invio di e-mail con Amazon SES e l'uso della libreria SDK.

3.7.2.1 Invio e-mail

All'interno del progetto Concierge Croccante l'invio di notifiche o messaggi via e-mail è stata semplificata realizzando una classe contenente al suo interno metodi appositi. Di tali metodi menzionati al punto 3.4.2 ne viene riportato il codice e analizzato il loro funzionamento:

```

1 const AWS = require('aws-sdk');
2 const ses = new AWS.SES({ apiVersion: '2010-12-01' });
3 class emailUtils {
4     static sendMail(from, to, message, oggetto) {
5         const params = {
6             Destination: { ToAddresses: [to] },
7             Message: {
8                 Body: {
9                     Html: { Charset: 'UTF-8', Data: message },
10                    Text: {
11                        Charset: 'UTF-8',
12                        Data: 'This is the message body in text format.'
13                    }
14                },
15                Subject: { Charset: 'UTF-8', Data: oggetto }
16            },
17            Source: from,
18            ReplyToAddresses: [from],
19        }
20        ses.sendEmail(params, (err, data) => {
21            if (err) {
22                console.log("[emailUtils.sendEmail error]:", err, err.stack)
23                throw err;
24            } else
25                console.log("[emailUtils.sendEmail]: Email notification sent!")
26        })
27    }
28 } module.exports = emailUtils;
  
```

Listing 3.9: Metodo sendMail

Il codice sopra riportato mostra il metodo `sendMail(from: var, to: var, message: var, oggetto: var)`: tale funzione non fa altro che inviare una e-mail con i campi from, to, message ed oggetto passati per parametro. La variabile costante `ses` non fa altro che richiamare il metodo fornito del pacchetto `aws-sdk` che si occuperà di richiamare le API necessarie all'utilizzo di AWS SES.

3.8 Lettura del Database

Funzionalità a supporto di quelle principali è la lettura del database DynamoDB. Quest’ultimo è popolato da una tabella contenente i nomi di tutti i dipendenti dell’azienda Crispy Bacon, i quali, vengono prelevati per essere utilizzati nelle fasi di controllo, come la ricerca di un dipendente per verificarne la sua esistenza. Il progetto predispone due classi adibite a questo tipo di funzionalità per interrogare il database: `databaseUtils.js` e `prepareQuery.js`.

```

1 const AWS = require("aws-sdk");
2 const prepareQuery = require("../utils/prepareQuery");
3
4 var instance;
5 class DataBaseUtils {
6     constructor() {
7         AWS.config.update({
8             region: "eu-west-1",
9             endpoint: "https://dynamodb.eu-west-1.amazonaws.com"
10        });
11        this.docClient = new AWS.DynamoDB.DocumentClient();
12    }
13
14    static getInstance() {
15        if (instance === undefined) {
16            instance = new DataBaseUtils();
17        }
18        return instance;
19    }
20
21    runQuery(params) {
22        return new Promise((resolve, reject) => {
23            this.docClient.query(params, function (err, data) {
24                if (err)
25                    reject(err);
26                else
27                    resolve(data);
28            })
29        })
30    }
31
32    runScan(params) { }
33    runPut(params) { }
34    static async chekEmployee(nome_dipendete) { }
35 }
36 module.exports = DataBaseUtils;

```

Listing 3.10: Metodo getInstance e runQuery

Il codice sopra riportato mostra la classe coi metodi per eseguire *query* di diversa tipologia: `runQuery(params: var)` non fa altro che richiamare il metodo presente nel pacchetto `aws-sdk` per l’esecuzione di un’interrogazione standard. Altri metodi come `runScan(params: var)` e `runPut(params: var)` eseguono una scansione e un inserimento nel database secondo i parametri²⁰ passati alle funzioni.

²⁰La classe `prepareQuery.js` svolge per l’appunto questo ruolo. I metodi della classe ritornano un oggetto contenente i parametri necessari per l’esecuzione della query desiderata

3.9 Async e Await

Nella la fase di stesura del codice durante il periodo di tirocinio è stato affrontato e studiato il tema della chiamate asincrone: Async/await functions. Questo modello di funzioni va inserito nel contesto delle *Promise* che, come suggerisce il suo nome, restituisce un risultato nel futuro che può essere di tipo *resolve*, nel caso di successo nell'azione svolta, o *reject* nel caso di un fallimento con conseguenza di errore. Per fare ciò la sintassi *async/await* permette di lavorare con le *Promise* in un modo più confortevole e facile da usare. Per spiegare il significato di questa sintassi vengono fatti i seguenti esempi estratti dal codice della Skill Concierge Croccante. Il seguente esempio mostra una funzione che ritorna una promessa, ovvero un risultato dopo aver eseguito determinati calcoli. In questo caso ritorna una Promise con *resolve* nel caso di un risultato da parte della chiamata API:

```

1  async function getIdMember(toTag) {
2      let tagged;
3      let members = await membersList();
4      let json_members = JSON.parse(members);
5
6      /* do something */
7
8      if (tagged === undefined)
9          tagged = false;
10
11     return tagged;
12 }
```

Listing 3.11: Esempio codice con Promise

Si necessita ora che la funzione chiamante il metodo `getIdMember(toTag: var)` attenda il risultato della *Promise* prima di proseguire con le istruzioni successive. Il modello *async/await* risolve tale necessità: dichiarando la funzione principale *async* e mettendo la keyword *await* davanti alla chiamata del metodo `getIdMember(toTag: var)` si attenderà il risultato ritornato da quest'ultima.

```

1  async function getIdMember(toTag) {
2      let tagged;
3      let members = await membersList();
4      let json_members = JSON.parse(members);
5
6      /* do something */
7
8      if (tagged === undefined)
9          tagged = false;
10
11     return tagged;
12 }
```

Listing 3.12: Esempio codice con Promise

Capitolo 4

Realizzazione e Testing

Capitolo 5

Conclusioni

Risultato ottenuto

- 5.1 Risultato ottenuto**
- 5.2 Analisi critica del prodotto**
- 5.3 Analisi critica del lavoro**
- 5.4 Valutazione degli strumenti utilizzati**
- 5.5 Possibili miglioramenti**
- 5.6 Possibili estensioni**

Appendice A

Appendice A

Citazione

Autore della citazione

Bibliografia

Riferimenti bibliografici

James P. Womack, Daniel T. Jones. *Lean Thinking, Second Editon*. Simon & Schuster, Inc., 2010.

Siti web consultati

Manifesto Agile. URL: <http://agilemanifesto.org/iso/it/>.