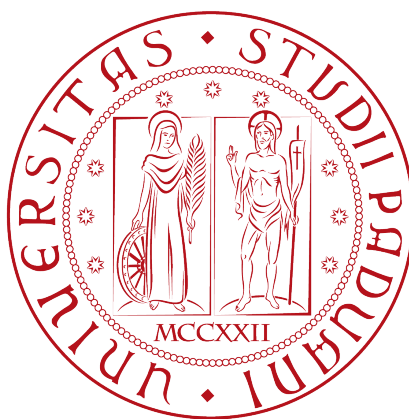


Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

CORSO DI LAUREA IN INFORMATICA



Integrazione di Amazon Rekognition e Lex  
all'interno di una web app basata su AWS  
Lambda

*Tesi di laurea triennale*

*Relatore*

Prof. Lamberto Ballan

*Laureanda*

Nicla Faccioli

---

ANNO ACCADEMICO 2021-2022



# Sommario

Il presente documento descrive l'attività di stage svolta presso l'azienda Zero12 s.r.l. Lo stage è stato svolto alla conclusione del percorso di studi della laurea triennale in Informatica ed ha avuto la durata di circa trecentoventi ore. L'obiettivo dello stage è stato l'integrazione dei servizi AWS di image recognition (Amazon Rekognition), automatic speech recognition e natural language understanding (Amazon Lex) all'interno di un'applicazione web serverless basata su AWS Lambda con lo scopo di facilitare l'inserimento dei dati.



“Let us light up the night, we shine in our own ways.  
Shine, dream, smile”

— 방탄소년단

## Ringraziamenti

*Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Lamberto Ballan, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.*

*Desidero ringraziare tutta la mia enorme famiglia ingrandita per avermi sempre sostenuta in ogni mia scelta e per avermi dato il coraggio necessario ad andare avanti. Grazie per tutti gli insegnamenti che mi avete trasmesso e che mi hanno permesso di raggiungere questo traguardo.*

*Un grazie speciale va alla mia mamma che, nonostante le difficoltà, gli imprevisti e il percorso dissestato che ho affrontato, mi ha sempre incoraggiata, ha creduto in me e ha saputo sdrammatizzare nei momenti che sembravano insuperabili. Grazie mamma per essere non solo una madre ma anche un'amica, sempre al mio fianco e pronta a farsi in quattro per aiutarmi a realizzare ogni mio sogno.*

*Grazie anche a papà per l'interesse che mostri ogni volta che ti racconto gli argomenti che studio, i progetti che realizzo o le cose che mi interessano. Grazie per esserci sempre quando ho bisogno di consigli o semplicemente di chiacchierare e condividere i miei pensieri.*

*Grazie a mio fratello Simone che con le sue chiamate serali interminabili non mi ha mai fatto sentire poi così lontana da casa. Grazie per avermi continuato a rendere partecipe di ogni parte della tua vita nonostante la distanza che ci separa e per essere così partecipe nella mia.*

*Vorrei ringraziare anche tutti i miei amici e le mie amiche per le mille avventure vissute in questi anni.*

*In particolar modo Anna che mi ha sopportato come coinquilina e compagna di stanza. Grazie per essere non solo la miglior coinquilina che si possa desiderare, ma anche una delle mie migliori amiche e una grande confidente. Grazie per essermi sempre stata vicina, per avermi ascoltato quando ne avevo bisogno e per aver condiviso con me tante risate, tanta ansia e anche qualche lacrima.*

*Un grazie anche a Giorgia che ha condiviso con me ogni giorno di vita universitaria sin dal principio. Grazie per tutte le avventure nate da idee pazzе che abbiamo avuto insieme e per essere sempre pronta a partire per qualsiasi luogo che ci venga in mente, da Banzai in centro a Padova fino al Giappone.*

Padova, Luglio 2022

Nicla Faccioli



# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	L'azienda . . . . .	1
1.2	L'offerta di stage . . . . .	1
1.3	Struttura del documento . . . . .	2
<b>2</b>	<b>Descrizione dello stage</b>	<b>3</b>
2.1	Introduzione al progetto . . . . .	3
2.2	Obiettivi formativi . . . . .	3
2.3	Requisiti . . . . .	3
2.4	Pianificazione . . . . .	5
<b>3</b>	<b>Tecnologie e strumenti</b>	<b>7</b>
3.1	Tecnologie per il back-end . . . . .	7
3.1.1	Serverless Framework . . . . .	7
3.1.2	Node.js . . . . .	7
3.1.3	AWS Lambda . . . . .	8
3.1.4	Amazon API Gateway . . . . .	8
3.1.5	Amazon DynamoDB . . . . .	8
3.1.6	Amazon S3 . . . . .	9
3.1.7	Amazon Rekognition . . . . .	9
3.1.8	Amazon Lex . . . . .	9
3.2	Tecnologie per il front-end . . . . .	10
3.2.1	TypeScript . . . . .	10
3.2.2	Angular . . . . .	10
3.2.3	Nebular . . . . .	10
3.3	Strumenti di supporto a progettazione e codifica . . . . .	11
3.3.1	Git . . . . .	11
3.3.2	AWS CodeCommit . . . . .	11
3.3.3	VisualStudio Code . . . . .	11
3.3.4	Balsamiq Wireframes . . . . .	12
<b>4</b>	<b>Descrizione dell'applicativo</b>	<b>13</b>
4.1	Architettura serverless . . . . .	13
4.1.1	Definizione delle resources . . . . .	13
4.1.2	Definizione delle funzioni Lambda . . . . .	15
4.1.3	Deploy del back-end . . . . .	15
4.2	Web app . . . . .	15
4.2.1	Funzionalità disponibili . . . . .	15

4.2.2	Deploy del front-end . . . . .	17
<b>5</b>	<b>Integrazione di Amazon Rekognition</b>	<b>19</b>
5.1	Presentazione del problema . . . . .	19
5.2	Progettazione . . . . .	19
5.2.1	Architettura . . . . .	19
5.2.2	Funzionamento generale . . . . .	22
5.2.3	Design dell'interfaccia . . . . .	23
<b>6</b>	<b>Integrazione di Amazon Lex</b>	<b>27</b>
6.1	Presentazione del problema . . . . .	27
6.2	Progettazione . . . . .	27
6.2.1	Architettura . . . . .	28
6.2.2	Funzionamento generale . . . . .	30
6.2.3	Design dell'interfaccia . . . . .	31
<b>7</b>	<b>Conclusioni</b>	<b>35</b>
7.1	Consuntivo finale . . . . .	35
7.2	Raggiungimento degli obiettivi . . . . .	36
7.3	Conoscenze acquisite e valutazione personale . . . . .	37
	<b>Glossary</b>	<b>39</b>
	<b>Bibliografia</b>	<b>41</b>



# Elenco delle figure

1.1	Logo di Zero12 s.r.l. . . . .	1
1.2	Logo dell'evento Stage-it 2022 . . . . .	2
3.1	Logo Serverless Framework . . . . .	7
3.2	Logo Node.js . . . . .	7
3.3	Logo AWS Lambda . . . . .	8
3.4	Logo Amazon API Gateway . . . . .	8
3.5	Logo Amazon DynamoDB . . . . .	8
3.6	Logo Amazon S3 . . . . .	9
3.7	Logo Amazon Rekognition . . . . .	9
3.8	Logo Amazon Lex . . . . .	10
3.9	Logo TypeScript . . . . .	10
3.10	Logo Angular . . . . .	10
3.11	Logo Nebular . . . . .	10
3.12	Logo Git . . . . .	11
3.13	Logo AWS CodeCommit . . . . .	11
3.14	Logo Visual Studio Code . . . . .	11
3.15	Logo Balsamiq . . . . .	12
4.1	Esempio del codice per la creazione di una tabella DynamoDB . . . . .	13
4.2	Esempio del codice per la creazione di un bucket S3 . . . . .	14
4.3	Esempio del codice per la creazione di una funzione Lambda . . . . .	15
4.4	Form per l'inserimento di una nuova partita . . . . .	16
4.5	Visualizzazione della predizione di una nuova partita . . . . .	16
4.6	Pagina per la visualizzazione dei punteggi Elo . . . . .	17
5.1	Snippet di codice per ottenere l'URL presigned . . . . .	20
5.2	Grafico del funzionamento del sistema di image recognition . . . . .	22
5.3	Schermata iniziale di calcetto . . . . .	23
5.4	Selezione dei giocatori . . . . .	24
5.5	Formazione squadre di calcetto . . . . .	24
5.6	Schermata iniziale di Duck Game . . . . .	25
5.7	Schermata iniziale di Mario Kart . . . . .	25
5.8	Selezione dei giocatori di Mario Kart . . . . .	26
6.1	Snippet di codice per ottenere l'URL presigned . . . . .	29
6.2	Grafico del funzionamento del chatbot . . . . .	31
6.3	Schermata iniziale con integrazione del chatbot . . . . .	32
6.4	Stato iniziale del pulsante di registrazione . . . . .	32

6.5	Pulsante nella fase di registrazione . . . . .	32
6.6	Pulsante nella fase di elaborazione . . . . .	33
6.7	Pulsante nella fase di riproduzione dell'audio . . . . .	33
6.8	Pulsante indica che il chatbot aspetta una risposta dall'utente . . . . .	33
6.9	Errore nell'elaborazione dell'audio . . . . .	33

## Elenco delle tabelle

2.1	Requisiti . . . . .	5
2.2	Pianificazione delle attività . . . . .	6
7.1	Consuntivo finale . . . . .	36
7.2	Tracciamento dei requisiti . . . . .	37
7.3	Completamento dei requisiti . . . . .	37

# Capitolo 1

## Introduzione

### 1.1 L'azienda

Zero12 S.r.l. è un'azienda informatica nata nel 2012 specializzata nello sviluppo di soluzioni cloud native, sempre in prima linea nel seguire l'evoluzione di questo paradigma tecnologico.

L'azienda è partner [AWS](#) e si occupa di progettazione e sviluppo software Web e Mobile per clienti provenienti da ambiti molto diversificati.

L'obiettivo di Zero12 S.r.l. è aiutare i propri clienti a definire percorsi di innovazione includendo le tecnologie più avanzate tra cui, per esempio, il cloud ed il *Machine Learning* per l'analisi di linguaggio naturale, immagini, video e per fare previsioni.



**Figura 1.1:** Logo di Zero12 s.r.l.

### 1.2 L'offerta di stage

Attualmente in azienda è presente una piattaforma denominata MariBa con lo scopo di registrare i risultati di gioco del personale a Mario Kart e calcetto balilla. L'inserimento di tali dati però è completamente manuale: ogni partita deve essere inizializzata con l'inserimento dei nickname di tutti i giocatori e, una volta conclusa, i risultati devono essere inseriti manualmente all'interno della piattaforma. L'idea dello stage è di semplificare l'inserimento di questi dati attraverso l'utilizzo di tecnologie [AWS](#) per il riconoscimento automatico dei giocatori e per la registrazione dei risultati comunicandoli vocalmente alla piattaforma.

Il progetto è stato proposto dall'azienda in occasione dell'evento Stage-it 2022 (logo in [Figura 1.2](#)) finalizzato all'incontro tra aziende e studenti.



**Figura 1.2:** Logo dell'evento Stage-it 2022

### 1.3 Struttura del documento

**Il secondo capitolo** descrive il progetto di stage e la pianificazione delle attività;

**Il terzo capitolo** definisce le tecnologie utilizzate durante lo stage;

**Il quarto capitolo** descrive il funzionamento generale dell'applicativo integrato;

**Il quinto capitolo** approfondisce lo sviluppo del sistema di image recognition;

**Il sesto capitolo** approfondisce lo sviluppo del sistema di voice service;

**Nel settimo capitolo** sono descritte le conclusioni dell'esperienza di stage e gli obiettivi raggiunti.

## Capitolo 2

# Descrizione dello stage

### 2.1 Introduzione al progetto

In Zero12 S.r.l. è stata creata una piattaforma denominata MariBa con lo scopo di registrare i risultati di gioco del personale a Mario Kart e calcetto balilla. Tale piattaforma è dotata di un sistema di intelligenza artificiale che, in base ai giocatori (o alle coppie nel caso del calcetto), è in grado di predire il risultato del match di gioco. Il limite della piattaforma attuale è che tutti i dati, dall'inizializzazione di una partita ai risultati finali, devono essere inseriti manualmente.

Al fine di rendere più immediato l'inserimento dei dati si vuole evolvere la piattaforma includendo le seguenti funzionalità:

- \* Sistema di *image recognition* per riconoscere i giocatori e definire i loro ruoli durante la fase di inizializzazione della partita e formazione delle squadre;
- \* Servizio vocale per l'inserimento dei risultati dei match giocati.

### 2.2 Obiettivi formativi

Gli obiettivi formativi dell'attività di stage sono i seguenti:

- \* Apprendere come sviluppare un applicativo web con controlli vocali;
- \* Apprendere come svolgere attività di integrazione con servizi di Machine Learning in ambito *image recognition*, [Automatic Speech Recognition](#) (ASR) e [Natural Language Understanding](#) (NLU);

### 2.3 Requisiti

Nel primo giorno di stage si è svolto un incontro con il tutor aziendale per definire in modo dettagliato i requisiti funzionali dell'applicativo, elencati in [Tabella 2.1](#).

Ogni requisito è classificato e identificato univocamente attraverso un codice composto secondo il seguente schema:

R[Priorità]-[Identificativo]

Dove:

- \* **R** indica che si tratta di un requisito;
- \* **Priorità** può assumere i seguenti valori:
  - **O**: requisito obbligatorio;
  - **D**: requisito desiderabile;
  - **F**: requisito facoltativo.
- \* **Identificativo**: numero progressivo che identifica il requisito in forma gerarchica, strutturato come segue:

[codicePadre].[codiceFiglio]

Codice	Descrizione
R0-1	Sviluppo della funzionalità di <i>face detection e recognition</i> per il riconoscimento automatico dei giocatori.
R0-1.1	L'utente deve poter utilizzare la webcam per scattare una fotografia ed effettuare il riconoscimento dei giocatori in modo automatico.
R0-1.2	L'utente deve poter associare un volto non riconosciuto ad un nickname esistente.
R0-1.3	L'utente deve poter creare un nuovo giocatore nel caso non fosse stato riconosciuto dal sistema ed egli non fosse ancora registrato.
R0-1.4	L'utente deve poter selezionare tra i volti individuati nella fotografia quali siano i giocatori che parteciperanno alla partita.
R0-1.5	L'utente deve poter scegliere la formazione delle squadre per le partite a calcetto balilla.
R0-1.6	L'utente deve poter utilizzare la funzionalità di predizione del risultato della partita anche in caso di riconoscimento automatico dei giocatori.
R0-2	Sviluppo di un <a href="#">chatbot</a> vocale per permettere l'utilizzo dell'applicativo web attraverso la pronuncia delle attività da svolgere.
R0-2.1	L'utente deve poter inizializzare una partita utilizzando il <a href="#">chatbot</a> .
R0-2.2	L'utente deve poter salvare i risultati di una partita utilizzando il <a href="#">chatbot</a> .
RD-1	Sviluppo di una pagina nel sito esistente per l'inserimento dei risultati per il gioco Duck Game.

RD-1.1	L'utente può inserire da due a otto giocatori per inizializzare una partita a Duck Game.
RD-1.2	L'utente può inserire i risultati di una partita a Duck Game e salvarli nel database.
RD-1.3	L'utente può richiedere la predizione del risultato finale della partita a Duck Game.
RD-2	Sviluppo di una versione semplificata della pagina per l'inserimento dei risultati delle partite a Mario Kart.
RF-1	Sviluppo di una <a href="#">skill</a> Alexa avente le stesse funzionalità del <a href="#">chatbot</a> vocale richiesto da R0-02.

Tabella 2.1: Requisiti

## 2.4 Pianificazione

La durata complessiva dello stage è stata di 8 settimane di lavoro a tempo pieno per un totale di circa trecentoventi ore.

Secondo il piano di lavoro iniziale definito con l'azienda, le attività sono distribuite come riportato in [Tabella 2.2](#).

Durata in ore	Settimana	Descrizione
40	1	* Studio delle tecnologie necessarie.
80	2, 3	* Progettazione e sviluppo di un micro-servizio per attività di <i>face detection</i> per la creazione di squadre di gioco; * Integrazione con la piattaforma esistente.
80	4, 5	* Progettazione e sviluppo di un micro-servizio per il controllo vocale; * Integrazione con la piattaforma esistente.

80	6, 7	<ul style="list-style-type: none"> <li>* Sviluppo della <a href="#">skill</a> Alexa per il controllo vocale e l'aggiornamento dei risultati;</li> <li>* Integrazione con la piattaforma esistente.</li> </ul>
40	8	<ul style="list-style-type: none"> <li>* Testing e stesura della documentazione di progetto delle attività di sviluppo condotte nelle settimane precedenti.</li> </ul>
<b>Totale ore:</b>		<b>320</b>

**Tabella 2.2:** Pianificazione delle attività

Durante il periodo di stage in azienda il piano di lavoro ha subito modifiche e di conseguenza il consuntivo delle attività svolte riportato nel [Capitolo 7](#) diverge dalla pianificazione qui presentata. Tali modifiche sono state effettuate in risposta a esigenze e richieste dell'azienda.

Durante tutta la durata del tirocinio sono stati effettuati stand-up giornalieri con il tutor aziendale per monitorare lo stato di avanzamento ed evidenziare eventuali problemi sorti.

Al termine dello stage si è svolta una presentazione del prodotto realizzato.



## Capitolo 3

# Tecnologie e strumenti

In questo capitolo vengono presentate le tecnologie utilizzate durante lo stage.

### 3.1 Tecnologie per il back-end

#### 3.1.1 Serverless Framework

*Serverless Framework* è un [framework](#) web che permette di costruire applicazioni [serverless](#) basate sul concetto [FaaS](#). Esso permette di definire funzioni Lambda e infrastrutture [AWS](#) utilizzando sintassi [YAML](#).



**Figura 3.1:** Logo Serverless Framework

#### 3.1.2 Node.js

*Node.js* è un ambiente runtime open source per l'esecuzione di codice *JavaScript* all'esterno di browser web. Esso consente infatti di utilizzare *JavaScript* come linguaggio di programmazione lato server. All'interno del progetto viene utilizzata la versione 12.x per compatibilità con il codice già presente.



**Figura 3.2:** Logo Node.js

### 3.1.3 AWS Lambda

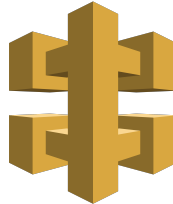
*AWS Lambda* è un servizio di calcolo serverless che permette l'esecuzione di codice per qualsiasi tipo di applicazione o servizio back-end senza bisogno di gestire un'infrastruttura server. *Lambda* gestisce le risorse di elaborazione scalando automaticamente in risposta alla potenza di calcolo richiesta. Il linguaggio utilizzato per lo sviluppo di funzioni *Lambda* è *Node.js v12.x*.



**Figura 3.3:** Logo AWS Lambda

### 3.1.4 Amazon API Gateway

*API Gateway* è un servizio Amazon che consente di creare *API RESTful* per permettere una comunicazione bidirezionale in tempo reale tra applicazioni e servizi di back-end. Le *API* definite nell'applicazione sviluppata sono state integrate alle rispettive funzioni *Lambda*.



**Figura 3.4:** Logo Amazon API Gateway

### 3.1.5 Amazon DynamoDB

*DynamoDB* è un database *NoSQL*, *serverless*, completamente gestito che supporta l'inserimento di dati di tipo chiave-valore. Facendo parte della famiglia di servizi messi a disposizione da Amazon, *DynamoDB* si integra senza difficoltà con tutti i servizi *AWS* e Amazon.

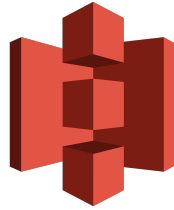


**Figura 3.5:** Logo Amazon DynamoDB

### 3.1.6 Amazon S3

*Amazon Simple Storage Service* (S3) è un servizio di archiviazione di oggetti, scalabile, sicuro e con ottime prestazioni. Al suo interno i dati sono organizzati in *bucket*. All'interno di ogni *bucket* è possibile definire dei prefissi per poter organizzare al meglio gli oggetti caricati.

All'interno del progetto questo servizio è stato utilizzato per effettuare l'hosting della web app e per il trasferimento indiretto di immagini e audio tra front-end e back-end.



**Figura 3.6:** Logo Amazon S3

### 3.1.7 Amazon Rekognition

*Amazon Rekognition* è un software *cloud-based* che mette a disposizione capacità di visione artificiale pre-addestrate e personalizzabili per estrarre informazioni dettagliate da immagini e video. Alcuni esempi di utilizzo sono la moderazione di contenuti e *sentiment analysis*.

All'interno del progetto è stato utilizzato per implementare la ricerca di volti all'interno di fotografie e per il loro riconoscimento in fase di inizializzazione di una partita e inserimento dei giocatori.



**Figura 3.7:** Logo Amazon Rekognition

### 3.1.8 Amazon Lex

*Amazon Lex* è un servizio di intelligenza artificiale completamente gestito che mette a disposizione modelli avanzati di linguaggio naturale. Questo permette di sviluppare interfacce di comunicazione all'interno di applicazioni software. Nel progetto è stato utilizzato per implementare un [chatbot](#) vocale in modo che l'utente potesse interagire con MariBa e registrare i risultati delle partite giocate.



**Figura 3.8:** Logo Amazon Lex

## 3.2 Tecnologie per il front-end

### 3.2.1 TypeScript

*TypeScript* è un linguaggio di programmazione sviluppato e mantenuto da Microsoft. Esso è un estensione del linguaggio di programmazione *JavaScript*: utilizza la stessa sintassi ma con l'aggiunta del supporto alla tipizzazione e alle interfacce.



**Figura 3.9:** Logo TypeScript

### 3.2.2 Angular

*Angular* è un [framework](#) open source sviluppato da Google. Esso permette lo sviluppo di applicazioni web organizzate in componenti attraverso l'utilizzo di *TypeScript*, [HTML](#) e [CSS](#).



**Figura 3.10:** Logo Angular

### 3.2.3 Nebular

*Nebular* è una libreria di *Angular* gratuita e open source per la creazione di interfacce utente.



**Figura 3.11:** Logo Nebular

## 3.3 Strumenti di supporto a progettazione e codifica

### 3.3.1 Git

*Git* è un sistema di [controllo di versione](#) distribuito. *Git* permette di tenere traccia di tutte le modifiche avvenute all'interno di un progetto o di un singolo file e associa a ciascuna di esse il relativo autore. Permette inoltre di tornare ad una versione precedente del software eliminando le modifiche effettuate successivamente allo stato desiderato. Tutto ciò rende più semplice la collaborazione tra sviluppatori nella stesura del codice durante la fase di sviluppo software.



Figura 3.12: Logo Git

### 3.3.2 AWS CodeCommit

*CodeCommit* è un servizio gestito, altamente scalabile e sicuro che consente l'hosting di *repository Git* privati. Esso custodisce i *repository* nel cloud [AWS](#) e supporta tutti i comandi *Git*. Si è scelto di utilizzare *CodeCommit* rispetto ad altri servizi equivalenti per compatibilità con la scelta aziendale e con il progetto esistente.



Figura 3.13: Logo AWS CodeCommit

### 3.3.3 VisualStudio Code

*Visual Studio Code* (VS Code) è un editor per il codice sorgente sviluppato da Microsoft. Esso possiede un controllo per *Git* integrato e mette a disposizione numerose estensioni per facilitare la stesura del codice. Un esempio è *Prettier*, estensione che automatizza la formattazione del codice in modo da mantenerlo ordinato e con uno stile consistente.



Figura 3.14: Logo Visual Studio Code

### 3.3.4 Balsamiq Wireframes

*Balsamiq Wireframes* è uno strumento grafico per la creazione di schizzi per interfacce utente e schermate (*wireframes*) di siti web e applicazioni. Durante lo stage è stata utilizzata la versione cloud. I *wireframes* creati sono stati revisionati dal tutor aziendale, il quale ha potuto inserire commenti sulle modifiche da apportare.



**Figura 3.15:** Logo Balsamiq

## Capitolo 4

# Descrizione dell'applicativo

In questo capitolo viene descritto l'applicativo le cui funzionalità sono state estese durante lo stage.

### 4.1 Architettura serverless

L'architettura della web app è basata sul *Serverless Framework* che, come suggerisce il nome, è un [framework](#) che permette di costruire architetture [serverless](#). Esso consente inoltre di definire l'architettura [AWS](#) (tabelle *DynamoDB*, funzioni *Lambda* e bucket *S3*) attraverso un file in formato [YAML](#).

Nei paragrafi seguenti vengono mostrati alcuni esempi della sintassi per la definizione delle infrastrutture necessarie ai nuovi servizi implementati ed il procedimento da seguire per effettuare il [deploy](#) della struttura [serverless](#).

#### 4.1.1 Definizione delle resources

Attraverso la direttiva *Resources* è possibile definire le risorse a cui le funzioni *Lambda* possono accedere, ovvero tabelle *DynamoDB* e bucket *S3*.

##### 4.1.1.1 Tabelle DynamoDB

```
610     duckRisultati:
611       Type: AWS::DynamoDB::Table
612       Properties:
613         TableName: ${self:service}-duck-risultati-${self:provider.stage}
614         AttributeDefinitions:
615           - AttributeName: id
616             AttributeType: "N"
617         KeySchema:
618           - AttributeName: id
619             KeyType: "HASH"
620         ProvisionedThroughput:
621           ReadCapacityUnits: 1
622           WriteCapacityUnits: 1
```

**Figura 4.1:** Esempio del codice per la creazione di una tabella DynamoDB

Nel codice sopra riportato:

- \* **Type** definisce il tipo di risorsa da creare;
- \* **TableName** indica il nome della tabella riportato nei servizi [AWS](#);
- \* **AttributeDefinitions** descrive gli attributi che compongono la chiave primaria;
- \* **KeySchema** definisce la struttura della chiave primaria. Nell'esempio la chiave è composta da un solo attributo ma *DynamoDB* permette di definire anche chiavi più complesse;
- \* **ProvisionedThroughput** specifica il numero di letture e scritture permesse della risorsa.

#### 4.1.1.2 Bucket S3

```

746 uploadbucket:
747   Type: AWS::S3::Bucket
748   Properties:
749     BucketName: ${self:service}-uploadbucket-${self:provider.stage}
750     AccessControl: Private
751     LifecycleConfiguration:
752       Rules:
753         - Id: audioExpiration
754           ExpirationInDays: 1
755           Prefix: "audio/"
756           Status: Enabled
757         - Id: imageExpiration
758           ExpirationInDays: 1
759           Prefix: "img/"
760           Status: Enabled
761     CorsConfiguration:
762       CorsRules:
763         - AllowedMethods:
764             - GET
765             - PUT
766             - POST
767             - HEAD
768           AllowedOrigins:
769             - "*"
770           AllowedHeaders:
771             - "*"

```

**Figura 4.2:** Esempio del codice per la creazione di un bucket S3

Nel codice sopra riportato:

- \* **Type** definisce il tipo di risorsa da creare;
- \* **BucketName** indica il nome del *bucket* riportato nei servizi [AWS](#);
- \* **AccessControl** specifica i permessi di accesso al *bucket*;
- \* **LifeCycleConfiguration** permette di definire delle regole per il ciclo di vita degli oggetti all'interno del bucket. Nel caso riportato sono state definite due regole per due diversi prefissi all'interno del *bucket* (*audioExpiration* e *imageExpiration*) entrambe con durata di un giorno;
- \* **CorsConfiguration** descrive la configurazione per le [CORS](#) per gli oggetti del *bucket*.



### 4.1.2 Definizione delle funzioni Lambda

Per definire le funzioni *Lambda* viene utilizzata la direttiva *functions*.

```
511 callRekognition:
512   handler: src/functions/rekognition/callRekognition.handler
513   events:
514     - http:
515       path: api/callRekognition
516       method: post
517       cors:
518         origin: "*"
519         headers:
520           - "*"

```

**Figura 4.3:** Esempio del codice per la creazione di una funzione Lambda

Nel codice sopra riportato:

- \* **handler** è il riferimento al file contenente il codice della funzione;
- \* **events** indica gli eventi che causano l'esecuzione della funzione *Lambda*. In particolare permette di specificare:
  - **http**: definisce gli *API Gateway* HTTP endpoint che, quando chiamati, provocano l'esecuzione della funzione;
  - **path**: definisce il percorso dell'endpoint e identifica la risorsa;
  - **method**: indica il tipo di accesso HTTP permesso;
  - **cors**: abilita le [CORS](#).

### 4.1.3 Deploy del back-end

Per effettuare il [deploy](#) delle modifiche alle funzioni e alle risorse definite nel file *serverless.yml* è sufficiente eseguire il comando

```
serverless deploy.
```

In alternativa, per fare il [deploy](#) di singole funzioni è possibile eseguire il comando

```
serverless deploy function -f <nome della funzione>.
```

## 4.2 Web app

La web app è realizzata utilizzando *Angular 12.x* in collaborazione con *Nebular*, libreria per la realizzazione di interfacce utente.

Nei paragrafi successivi vengono descritte le funzionalità disponibili all'interno dell'applicativo, in parte già presenti prima dell'inizio del tirocinio. Viene inoltre mostrato come effettuare il [deploy](#) delle modifiche effettuate al front-end.

### 4.2.1 Funzionalità disponibili

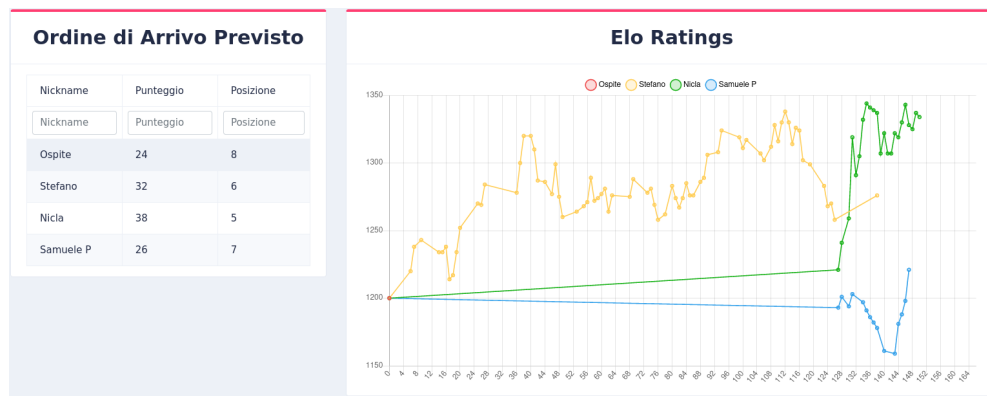
Salvataggio dei risultati

All'interno dell'applicativo era già presente la possibilità di inserire i risultati delle partite giocate a due giochi differenti: Mario Kart e Calciotto. Durante lo stage, l'applicativo è stato esteso includendo la possibilità di caricare i risultati anche per un terzo gioco: Duck Game. Tutti i risultati inseriti vengono memorizzati all'interno di tabelle *DynamoDB* dedicate e successivamente utilizzati per aggiornare il punteggio [Elo](#) dei giocatori che hanno partecipato.

**Figura 4.4:** Form per l'inserimento di una nuova partita

### Predizione dei risultati di nuove partite

Nell'applicativo viene utilizzato un algoritmo di *Machine Learning* che, sfruttando i punteggi [Elo](#) dei giocatori partecipanti, è in grado di predire i risultati della partita. Ogni volta che vengono inserite nuove partite, l'algoritmo migliora le proprie predizioni diventando sempre più preciso.



**Figura 4.5:** Visualizzazione della predizione di una nuova partita

### Visualizzazione delle statistiche

Per ciascun gioco supportato all'interno della web app è presente la pagina [Elo rating](#) che permette di visualizzare i punteggi [Elo](#) di tutti i giocatori. Inoltre, all'interno della stessa pagina, viene mostrato un grafico con lo storico di tutti i punteggi per i dieci giocatori più frequenti nello specifico gioco scelto.



**Figura 4.6:** Pagina per la visualizzazione dei punteggi Elo

### 4.2.2 Deploy del front-end

Per eseguire il [deploy](#) delle modifiche al front-end è necessario seguire i seguenti passi:

1. Nella cartella del progetto, generare la cartella *dist* eseguendo il comando

```
ng build
```

2. Utilizzare la console [AWS](#) per visualizzare il contenuto del *bucket* in cui è stato eseguito l'hosting del sito;
3. Sostituire il contenuto del *bucket* con i file presenti nella cartella *dist* generata;
4. Utilizzare il servizio [CloudFront](#) per creare un'*invalidation* e comunicare che i contenuti del sito sono cambiati. In questo modo il servizio renderà visibili le modifiche effettuate.



## Capitolo 5

# Integrazione di Amazon Rekognition

In questo capitolo viene approfondito lo sviluppo e l'integrazione del sistema di image recognition all'interno dell'applicativo.

### 5.1 Presentazione del problema

MariBa è una piattaforma per il salvataggio dei risultati delle partite giocate dai dipendenti di Zero12 S.r.l. durante le pause pranzo e caffè.

L'inizializzazione di una partita prevede l'inserimento dei nomi di tutti i giocatori, specificando anche la posizione (attacco o difesa) e le squadre nel caso del biliardino. Questo procedimento di inserimento dei dati iniziali risultava laborioso ed è stata quindi espressa la necessità di velocizzare tale procedura.

L'idea è quella di scattare una foto ai giocatori partecipanti in modo tale che l'applicazione li riconosca in modo automatico.

### 5.2 Progettazione

In questa sezione vengono descritti l'architettura ed il funzionamento del sistema di riconoscimento facciale implementato.

Successivamente viene presentato e approfondito il design dell'applicazione web dopo l'integrazione.

#### 5.2.1 Architettura

Il micro-servizio sviluppato fa uso di diverse tecnologie rese disponibili da [AWS](#):

- \* *Amazon Rekognition*;
- \* *Amazon S3*;
- \* *AWS Lambda*

Nei prossimi paragrafi vengono descritte nel dettaglio tali tecnologie, il loro ruolo all'interno dell'applicativo e come esse collaborino tra loro per il raggiungimento dell'obiettivo.

#### 5.2.1.1 Amazon Rekognition

*Amazon Rekognition* è un software *cloud-based* che mette a disposizione capacità di visione artificiale pre-addestrate e personalizzabili per estrarre informazioni dettagliate da immagini e video. Nel caso del progetto è stato utilizzato per indicizzare le facce e permetterne quindi il riconoscimento. In particolare, tutte le facce sono state salvate all'interno di una raccolta (*collection*) ed a ciascuna di esse è stato assegnato un ID univoco (*faceId*). Per effettuare un riconoscimento si procede ad una ricerca di eventuali corrispondenze all'interno di tale *collection*.

Di seguito sono elencate e descritte le specifiche funzioni di *Rekognition* utilizzate:

- \* **DetectFaces**: individua le cento facce di dimensione maggiore presenti nell'immagine. Per ogni viso individuato ne restituisce varie informazioni, in particolare la *bounding box*;
- \* **IndexFaces**: individua i visi all'interno di un'immagine e li aggiunge ad una *collection* specificata. Per questioni di sicurezza *Rekognition* non salva direttamente l'immagine contenente la faccia ma salva solamente alcune informazioni sulle caratteristiche che ne permettano il riconoscimento.
- \* **SearchFacesByImage**: data un'immagine, vengono identificate le facce presenti e successivamente ne vengono cercate delle corrispondenze all'interno di una *collection* specificata.

#### 5.2.1.2 S3

*Amazon Simple Storage Service* (S3) è un servizio di archiviazione oggetti. Al suo interno i dati sono organizzati in *bucket*. All'interno di ogni *bucket* è possibile definire dei prefissi per poter organizzare al meglio gli oggetti caricati (simile al concetto di "cartella").

Per evitare un passaggio diretto delle immagini tra front-end e back-end è stato utilizzato un *bucket*. S3 infatti fornisce la possibilità di generare un URL per effettuare operazioni di upload o download in una specifica posizione all'interno di un *bucket* senza necessità di autenticazione. In particolare viene effettuata la seguente operazione per la generazione di tale URL:

```
9      const params = {
10        Bucket: process.env.UPLOAD_BUCKET,
11        Key: data.fileName,
12        Expires: 3600,
13        ContentType: data.fileType,
14      };
15
16      const signedUrl = await s3.getSignedUrl("putObject", params);
```

**Figura 5.1:** Snippet di codice per ottenere l'URL presigned

Nel codice sopra riportato:

- \* **Bucket**: nome del *bucket* su cui effettuare l'operazione;

- \* **Key**: nome del file che si vuole caricare con il presigned che si ottiene, specificando eventuali prefissi;
- \* **ContentType**: tipo del file che si vuole caricare. Nel caso delle immagini caricate da MariBa è *image/jpeg*;
- \* **“putObject”**: tipo di operazione che si vuole effettuare con l’URL generato. In questo caso si tratta di un’operazione PUT.

Le immagini caricate hanno utilità molto breve: infatti una volta che il back-end ne ha effettuato il download, esse non verranno più utilizzate. Di conseguenza, per evitare di occupare spazio all’interno del *bucket* inutilmente, è stata impostata una *lifecycle rule* in modo tale che tutte le immagini (con prefisso *img/*) vengano eliminate in modo automatico dopo un giorno dal loro caricamento. Per informazioni specifiche riguardo alla *lifecycle rule* fare riferimento alla [sottosottosezione 4.1.1.2](#).

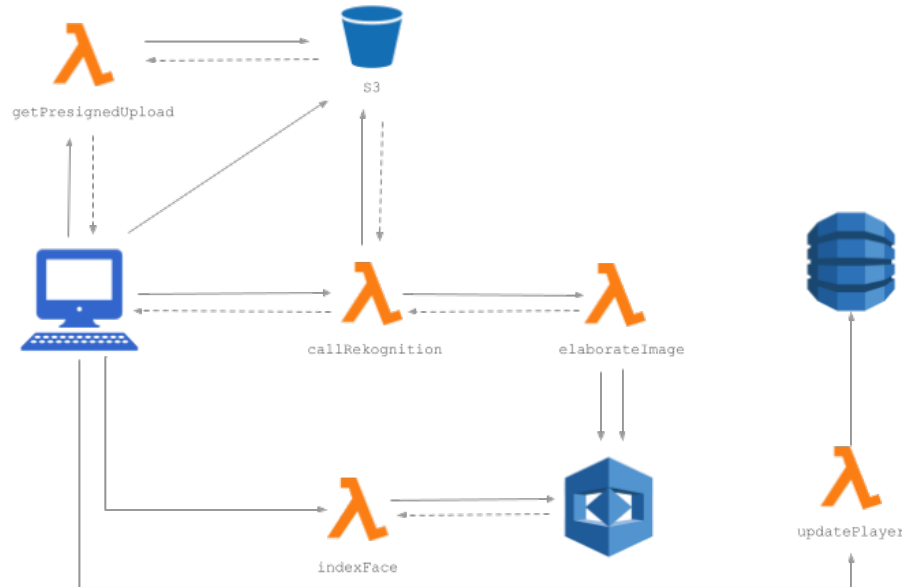
### 5.2.1.3 AWS Lambda

Di seguito sono descritte le funzioni *Lambda* implementate per il corretto funzionamento della funzionalità di riconoscimento automatico dei giocatori:

- \* **getPresignedUpload**: richiede ad *S3* il presigned URL per il caricamento dell’immagine in uno specifico *bucket* e con uno specifico prefisso;
- \* **callRekognition**: effettua il download dell’immagine dal *bucket* *S3* in modo che possa essere utilizzata da **elaborateImage**;
- \* **elaborateImage**: effettua le chiamate effettive a *Rekognition*. In particolare:
  1. Individua i volti nella fotografia utilizzando **rekognition.DetectFaces**, la quale restituisce le relative *bounding box*;
  2. Ritaglia i volti individuati utilizzando tali *bounding box*;
  3. Per ciascun volto ritagliato chiama **rekognition.SearchFacesByImage** per cercare eventuali corrispondenze all’interno della *collection* contenente tutti i volti precedentemente salvati;
  4. Se sono state trovate delle somiglianze, **rekognition.SearchFacesByImage** restituisce i *faceId* corrispondenti in ordine decrescente di probabilità;
  5. Per ciascun volto individuato nell’immagine iniziale, **elaborateImage** restituisce le informazioni sulla relativa *bounding box*. Inoltre, nel caso fossero state trovate corrispondenze, restituisce anche il *faceId*. In caso contrario restituisce invece i crop dei visi non riconosciuti.
- \* **updatePlayer**: aggiorna un giocatore già presente nel database con il nuovo *faceId* nel momento in cui esso viene associato dall’utente;
- \* **indexFace**: utilizza **rekognition.IndexFaces** per indicizzare nuove facce all’interno della *collection* di *Rekognition*.

### 5.2.2 Funzionamento generale

Il funzionamento generale del sistema di riconoscimento tramite immagine è schematizzato in [Figura 5.2](#).



**Figura 5.2:** Grafico del funzionamento del sistema di image recognition

In particolare avvengono le seguenti operazioni:

- \* Il front-end utilizza la funzione `getPresignedUpload` per ottenere l'URL per il caricamento dell'immagine nel *bucket* *S3*;
- \* Una volta ottenuto il link, il front-end procede al caricamento;
- \* Chiama la funzione `callRekognition` e attende i dati delle facce individuate e/o riconosciute;
- \* `callRekognition` scarica l'immagine dal *bucket* e la passa a `elaborateImage`;
- \* `elaborateImage` esegue le chiamate effettive a *Rekognition* come descritte nella [sottosottosezione 5.2.1.1](#);
- \* `elaborateImage` ritorna il risultato dell'elaborazione opportunamente strutturato a `callRekognition` che a sua volta lo restituisce al front-end;
- \* Il front-end visualizza i dati ricevuti;
- \* Se viene richiesto di registrare un nuovo utente:
  - Viene chiamata `indexFace` per indicizzare il crop del viso non ancora registrato nella *collection* di *Rekognition*;
  - `indexFace` restituisce il *faceId* del viso indicizzato;
  - Viene aggiunto un nuovo record nel database contenente le informazioni del nuovo giocatore, compreso il *faceId* ricevuto;



- \* Se viene richiesto di associare un nickname già esistente ad un viso:
  - Viene chiamata `indexFace` per indicizzare il crop del viso non ancora registrato nella *collection* di *Rekognition*;
  - `indexFace` restituisce il *faceId* del viso indicizzato;
  - Viene chiamata `updatePlayer` per inserire il *faceId* nel record del giocatore da associare. Nel caso in cui vi fosse già un *faceId*, questo viene sostituito.

### 5.2.3 Design dell'interfaccia

Per il design dell'interfaccia, prima della sua effettiva codifica, sono stati realizzati dei *wireframes* utilizzando *Balsamiq*. In questo modo si è potuto definire il flusso di funzionamento a livello di front-end.

Successivamente, lo sviluppo dell'interfaccia è avvenuto utilizzando *Angular 12.x* in collaborazione con la libreria *Nebular*, specifica per la creazione di interfacce utente.

Nei paragrafi seguenti viene mostrata l'interfaccia realizzata per l'inserimento dei giocatori attraverso l'utilizzo della funzionalità di riconoscimento.

#### 5.2.3.1 Inserimento di una nuova partita di calcetto

Nella pagina per l'inizializzazione di una nuova partita di calcetto è stato aggiunto un pulsante per attivare la webcam e scattare la foto contenente i volti dei giocatori (Figura 5.3). Lo scatto viene quindi inviato a *Amazon Rekognition* per effettuare il riconoscimento facciale.

The screenshot shows a web interface for setting up a soccer game. At the top is a green bar with the word 'Partita'. Below this, there are two main sections for 'Squadra Rossa' (red) and 'Squadra Blu' (blue). Each section contains two dropdown menus: 'Difensore' and 'Attaccante'. Below these sections, there is a button labeled 'oppure' followed by a blue button with a camera icon and the text 'USA FOTOCAMERA'. At the bottom, there is a grey button labeled 'PREDIZIONE RISULTATO'.

Figura 5.3: Schermata iniziale di calcetto

Una volta ricevuto il risultato dell'elaborazione, i giocatori vengono visualizzati in card selezionabili per la formazione delle squadre (Figura 5.4). I giocatori possono appartenere a tre categorie differenti:

- \* **Giocatore riconosciuto:** viene mostrato il nickname corrispondente al volto riconosciuto;
- \* **Giocatore non riconosciuto ma registrato:** il giocatore è già registrato ma non ha ancora un volto associato. Quando selezionato richiede l'associazione del nickname al viso scegliendo tra i nickname già presenti nel database;
- \* **Giocatore non riconosciuto e non registrato:** quando selezionato richiede la registrazione di un nuovo giocatore;

Nel caso in cui uno o più giocatori partecipanti non fossero presenti all'interno della fotografia scattata o non fossero stati individuati, è possibile inserirli manualmente: premendo sull'apposito pulsante viene mostrato un pop-up nel quale selezionare il nickname del giocatore da aggiungere. Una volta confermato l'inserimento, viene mostrata la card corrispondente.

Il numero di giocatori selezionabili per ciascuna squadra è esattamente due.

**Figura 5.4:** Selezione dei giocatori

Dopo la selezione dei giocatori sarà infine possibile scambiare i due nickname all'interno di ciascuna squadra per associare loro il ruolo desiderato e creare la formazione definitiva (Figura 5.5).

**Figura 5.5:** Formazione squadre di calcetto

### 5.2.3.2 Inserimento di una nuova partita di Mario Kart e Duck Game

Il procedimento da seguire per l'inserimento di una partita di Mario Kart o di Duck Game tramite l'utilizzo del riconoscimento facciale dei giocatori è pressoché il medesimo. La sola differenza tra i due giochi è individuabile nel numero di giocatori selezionabili:

- \* per Mario Kart devono essere inseriti esattamente quattro giocatori (Figura 5.7);
- \* per Duck Game si possono inserire da un minimo di due ad un massimo di otto giocatori (Figura 5.6).

Ad entrambe le schermate, come per calcetto, è stato quindi aggiunto un pulsante per attivare la webcam e scattare la fotografia da inviare ad *Amazon Rekognition*.

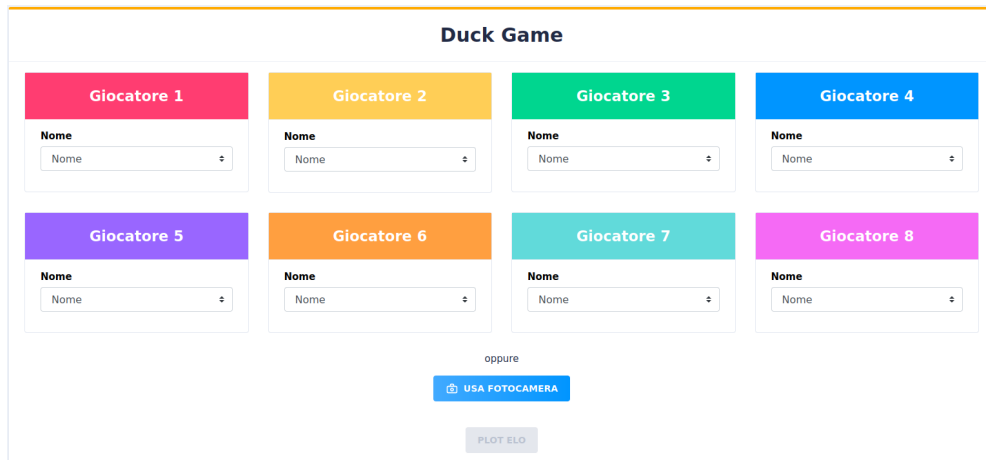


Figura 5.6: Schermata iniziale di Duck Game

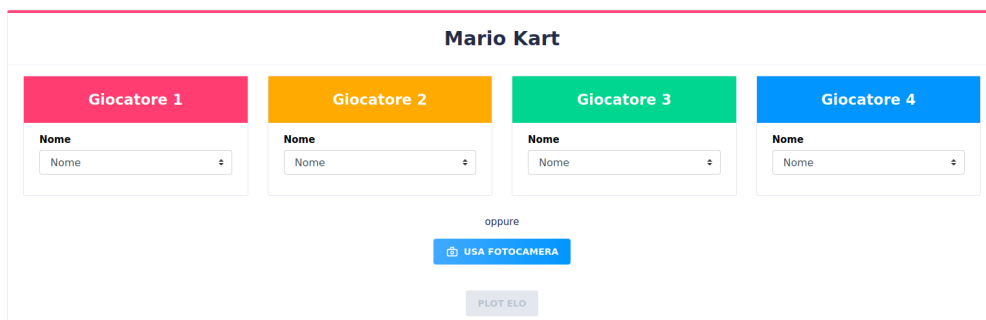


Figura 5.7: Schermata iniziale di Mario Kart

Una volta ricevuto il risultato dell'elaborazione, i giocatori vengono visualizzati in card selezionabili (Figura 5.8). I giocatori possono appartenere a tre categorie differenti:

- \* **Giocatore riconosciuto:** viene mostrato il nickname corrispondente al volto riconosciuto;
- \* **Giocatore non riconosciuto ma registrato:** il giocatore è già registrato ma non ha ancora volto associato. Quando selezionato richiede l'associazione del nickname al viso scegliendo tra i nickname già presenti nel database;
- \* **Giocatore non riconosciuto e non registrato:** quando selezionato richiede la registrazione di un nuovo giocatore;

Nel caso in cui uno o più giocatori partecipanti non fossero presenti all'interno della fotografia scattata o non fossero stati individuati, è possibile inserirli manualmente: premendo sull'apposito pulsante viene mostrato un pop-up nel quale selezionare il nickname del giocatore da aggiungere. Una volta confermato l'inserimento, viene

mostrata la card corrispondente.

The screenshot shows a web interface titled "Mario Kart" for player selection. It features three main cards, each with a green checkmark icon in its top-left corner. The first card, titled "Nicla", displays a photo of a woman. The second card, titled "Sono:", shows a blurred photo and includes a dropdown menu, an "OK" button, and the word "oppure" (or). Below this card is a blue button labeled "CREA NUOVO GIOCATORE". The third card, titled "Ospite", shows a grey circle icon and includes a blue button labeled "+ AGGIUNGI MANUALMENTE" and a grey button labeled "PLOT ELO". A modal dialog box is open over the "Ospite" card, containing a "Nickname" dropdown menu and an "OK" button.

**Figura 5.8:** Selezione dei giocatori di Mario Kart

## Capitolo 6

# Integrazione di Amazon Lex

In questo capitolo viene approfondito lo sviluppo e l'integrazione del [chatbot](#) vocale per l'interazione tra l'utente e l'applicativo.

### 6.1 Presentazione del problema

MariBa è una piattaforma per il salvataggio dei risultati delle partite giocate dai dipendenti di Zero12 S.r.l. durante le pause pranzo e caffè.

Il salvataggio di una partita conclusa prevede l'inserimento di molti dati contenenti le informazioni rilevanti alle predizioni delle partite successive. In particolare viene richiesto:

\* **Calcetto:**

- Autogol effettuati da ciascun giocatore (4 campi da compilare);
- Gol effettuati da ciascun giocatore (4 campi da compilare);

\* **Mario Kart:**

- Torneo giocato (1 campo da compilare);
- Posizione finale di ciascun giocatore (4 campi da compilare);
- Punteggio finale di ciascun giocatore (4 campi da compilare).

\* **Duck Game:**

- Punteggio di ciascun giocatore (da 2 a 8 campi da compilare).

Poiché l'inserimento risultava laborioso, è stato richiesto di implementare un [chatbot](#) vocale che permettesse il salvataggio delle partite giocate.

### 6.2 Progettazione

In questa sezione vengono descritti l'architettura ed il funzionamento del sistema di interazione vocale implementato.

Inoltre viene presentato il design dell'applicazione web dopo l'integrazione.

### 6.2.1 Architettura

Il micro-servizio sviluppato fa uso dei seguenti servizi messi a disposizione da [AWS](#):

- \* *Amazon Lex v2*;
- \* *Amazon S3*;
- \* *AWS Lambda*.

#### 6.2.1.1 Amazon Lex v2

*Amazon Lex* è un servizio di intelligenza artificiale completamente gestito che mette a disposizione modelli avanzati di linguaggio naturale per permettere lo sviluppo di interfacce di comunicazione all'interno di applicazioni software. Nel progetto di stage è stata utilizzata la versione due di *Amazon Lex* per implementare un [chatbot](#) vocale per interagire con l'applicazione e registrare i risultati delle partite giocate.

La creazione di [chatbot](#) con *Amazon Lex* può essere effettuata attraverso l'utilizzo della console messa a disposizione da [AWS](#).

Per ogni [chatbot](#) di *Lex* possono essere definiti diversi *intent*, ovvero azioni che si vuole portare a termine attraverso l'utilizzo dello stesso. In ciascun *intent* possono essere definiti:

- \* **Fraasi d'esempio:** frasi che l'utente deve pronunciare al [chatbot](#) per richiamare uno specifico *intent*. Nella console vengono inserite alcune frasi rappresentative e successivamente *Lex* sarà in grado di comprenderne delle varianti;
- \* **Slot:** variabili a cui il [chatbot](#) assegnerà un valore sulla base dell'input fornitogli dall'utente. Gli slot sono tipizzati: *Lex* mette a disposizione alcuni tipi built-in ma permette anche di definire tipi personalizzati;
- \* **Flusso della conversazione:** con quale ordine chiedere all'utente il valore degli *slot* necessari e quali frasi usare per farlo.

Inoltre, per effettuare delle validazioni sull'input, *Lex* include la possibilità di associare al [chatbot](#) una funzione *Lambda* da richiamare alla fine di un *intent* oppure ogni volta che viene fornito un valore per uno *slot*.

Infine, per la realizzazione del prodotto, sono state utilizzate le seguenti funzioni di *Amazon Lex*:

- \* **RecognizeUtterance:** invia l'input (audio o testo) dell'utente ad *Amazon Lex*. Quest'ultimo interpreta l'input utilizzando i modelli di machine learning costruiti per il [chatbot](#);
- \* **PutSession:** permette di modificare lo stato della sessione creata quando un utente inizia la conversazione con il [chatbot](#). Tale sessione è identificata attraverso il *sessionId* (assegnato automaticamente o specificato manualmente).

#### 6.2.1.2 Amazon S3

*Amazon Simple Storage Service* (S3) è un servizio di archiviazione oggetti. Al suo interno i dati sono organizzati in *bucket*. All'interno di ogni *bucket* è possibile definire

dei prefissi per poter organizzare al meglio gli oggetti caricati (simile al concetto di “cartella”).

Per evitare un passaggio diretto degli audio tra front-end e back-end si è utilizzato un *bucket*. S3 infatti fornisce la possibilità di generare un URL per effettuare operazioni di upload o download in una specifica posizione all’interno del *bucket* senza necessità di autenticazione.

In particolare, per generare l’URL per l’upload viene effettuata la seguente operazione:

```
9      const params = {
10        Bucket: process.env.UPLOAD_BUCKET,
11        Key: data.fileName,
12        Expires: 3600,
13        ContentType: data.fileType,
14      };
15
16      const signedUrl = await s3.getSignedUrl("putObject", params);
```

**Figura 6.1:** Snippet di codice per ottenere l’URL presigned

Nel codice sopra riportato:

- \* **Bucket**: nome del *bucket* su cui effettuare l’operazione;
- \* **Key**: nome del file che si vuole caricare con il presigned che si ottiene, specificando eventuali prefissi;
- \* **ContentType**: tipo del file che si vuole caricare. Nel caso degli audio registrati da MariBa, il formato è *audio/wav*;
- \* **“putObject”**: tipo di operazione che si vuole effettuare con l’URL generato. In questo caso si tratta di un’operazione PUT.

Gli audio caricati hanno utilità molto breve: infatti una volta che il back-end ne ha effettuato il download, essi non verranno più utilizzati. Di conseguenza, per evitare di occupare spazio all’interno del *bucket* inutilmente, è stata impostata una *lifecycle rule* in modo tale che tutte gli audio (con prefisso *audio/*) vengano eliminati in modo automatico dopo un giorno dal loro caricamento.

L’operazione eseguita per generare l’URL per il download è molto simile a quella sopra riportata con la sola differenza che “*putObject*” viene sostituito con “*getObject*”.

### 6.2.1.3 AWS Lambda

Di seguito sono descritte le funzioni *Lambda* implementate per il funzionamento del *chatbot Lex*:

- \* **getPresignedUpload**: richiede il presigned URL per il caricamento dell’audio;
- \* **elaborateAudio**: effettua la chiamata effettiva a *Lex* specificando il *chatbot* da utilizzare. In particolare:
  1. Scarica l’audio dell’utente da *S3*;
  2. Chiama `lex.RecognizeUtterance` sull’audio ottenuto;
  3. `lex.RecognizeUtterance` restituisce l’audio contenente la risposta del *chatbot*;

4. Carica l'audio di risposta su *S3*;
5. Restituisce al front-end le informazioni sulla posizione nel bucket dell'audio di risposta.

\* **maribaBot**: funzione utilizzata dal bot per la validazione dell'input e/o alla conclusione di un intent.

In particolare, la funzione controlla che i giocatori inseriti dall'utente siano effettivamente presenti all'interno del database:

- Poiché i nickname sono una trascrizione dell'audio ricevuto dall'utente, è stata utilizzata la [distanza di Levenshtein](#) per fare in modo che non fosse necessaria una corrispondenza perfetta con il nickname salvato;
- Quando viene effettuato tale controllo, viene tenuto come definitivo il nickname con [distanza di Levenshtein](#) minore;
- Nel caso in cui non fossero presenti nel database nickname con [distanza di Levenshtein](#) minore di tre rispetto a quello inserito, il nickname viene considerato come non valido e lo slot corrispondente viene chiesto nuovamente all'utente.

Inoltre **maribaBot** utilizza la funzione **putSession** di *Lex* per creare i *session attributes* (attributi di sessione) all'inizio di ciascuna sessione. Tali attributi sono necessari per la memorizzazione dei dati comunicati al **chatbot** dall'utente e verranno utilizzati alla conclusione della conversazione per il salvataggio della partita.

\* **getPresignedDownloadAudio**: richiede l'URL per il download dell'audio da *S3*.

### 6.2.2 Funzionamento generale

Nel **chatbot** implementato durante lo stage sono stati definiti diversi *intent*. Infatti, per ciascun gioco disponibile su MariBa sono stati implementati due *intent*:

- \* Uno per l'inserimento dei giocatori;
- \* Uno per l'inserimento dei risultati della partita.

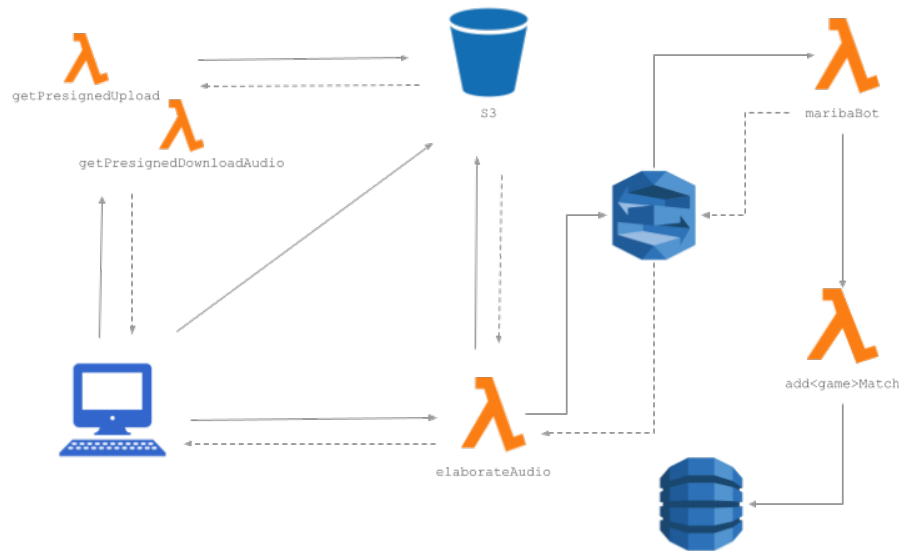
Inoltre, sono stati definiti i seguenti slot:

- \* Uno per ciascun giocatore;
- \* Uno per ciascun dato richiesto nel salvataggio della partita nell'interfaccia grafica di MariBa (punteggi, posizioni e nome del torneo di Mario Kart).

Pronunciando ad esempio “Voglio registrare una partita a Mario Kart” il **chatbot** capirà l'intenzione dell'utente e inizierà chiedendo quali siano i giocatori partecipanti. Una volta concluso l'inserimento dei nominativi, verrà richiamato automaticamente il secondo *intent* per permettere l'inserimento dei risultati.

Il funzionamento generale dell'applicazione è riportato nella [Figura 6.2](#) e descritto di seguito.





**Figura 6.2:** Grafico del funzionamento del chatbot

1. L'utente registra un audio attraverso l'apposita interfaccia all'interno del sito MariBa;
2. Il front-end utilizza la funzione `getPresignedUpload` per ottenere l'URL per il caricamento dell'audio nel *bucket* di *S3*;
3. Una volta ottenuto il link, il front-end procede al caricamento;
4. Chiama la funzione `elaborateAudio` e attende il completamento della sua esecuzione;
5. `elaborateAudio` scarica l'audio dal *bucket* e lo invia a *Lex* per l'elaborazione;
6. *Lex* utilizza `maribaBot` per la validazione delle informazioni estratte dall'audio e restituisce a sua volta un audio contenente la risposta generata dal chatbot;
7. `elaborateAudio` carica la risposta su *S3* e restituisce le informazioni sulla posizione dell'oggetto all'interno del *bucket* al front-end;
8. Il front-end chiama la funzione `getPresignedDownloadAudio` per ottenere l'URL per il download dell'audio;
9. L'audio viene scaricato e viene riprodotto;
10. Quando viene confermato il salvataggio di una partita, `maribaBot` procede al caricamento nel database dei dati estrapolati dalla conversazione con l'utente utilizzando la corretta funzione *Lambda* a seconda del gioco utilizzato.

### 6.2.3 Design dell'interfaccia

Per il design dell'interfaccia, prima della sua effettiva codifica, sono stati realizzati dei *wireframes* utilizzando *Balsamiq*. In questo modo si è potuto definire il flusso di

funzionamento a livello di front-end.

Successivamente, la realizzazione dell'interfaccia è avvenuta utilizzando *Angular 12.x* in collaborazione con la libreria *Nebular*, specifica per lo sviluppo di interfacce utente.

Per l'utilizzo del *chatbot* vocale è stato integrato all'interno dell'header del sito un pulsante. Tenendo premuto tale pulsante è possibile registrare l'audio contenente le richieste desiderate. Per aiutare l'utente sono stati inseriti alcuni *tooltip* contenenti consigli su frasi da pronunciare o su come iniziare una registrazione.



**Figura 6.3:** Schermata iniziale con integrazione del chatbot

L'aspetto del pulsante varia a seconda dello stato della conversazione:

#### Stato iniziale

L'applicativo è pronto per l'inizio di una nuova conversazione.



**Figura 6.4:** Stato iniziale del pulsante di registrazione

#### In registrazione

L'applicativo sta registrando l'audio.



**Figura 6.5:** Pulsante nella fase di registrazione

**Elaborazione dell'audio**

L'applicativo sta procedendo all'invio dell'audio per l'elaborazione.



**Figura 6.6:** Pulsante nella fase di elaborazione

**Riproduzione dell'audio ricevuto da Lex**

L'applicativo sta riproducendo l'audio ricevuto in risposta dal [chatbot](#).



**Figura 6.7:** Pulsante nella fase di riproduzione dell'audio

**Pronto per la registrazione della risposta**

L'applicativo è pronto per la registrazione da parte dell'utente della prossima interazione con il [chatbot](#).



**Figura 6.8:** Pulsante indica che il chatbot aspetta una risposta dall'utente

**Stato di errore**

In caso di errore nell'elaborazione dell'audio viene mostrato a schermo un *pop-up*.



**Figura 6.9:** Errore nell'elaborazione dell'audio



## Capitolo 7

# Conclusioni

### 7.1 Consuntivo finale

Durata in ore	Settimana	Descrizione
40	1	* Studio delle tecnologie necessarie.
80	2, 3	* Progettazione e sviluppo di un micro-servizio per attività di <i>face detection</i> per la creazione di squadre di gioco; * Integrazione con la piattaforma esistente.
40	4	* Progettazione e sviluppo di una sezione del sito per l'inserimento dei risultati di un nuovo gioco; * Integrazione con la piattaforma esistente.
100	5, 6, 7	* Progettazione e sviluppo di un micro-servizio per il controllo vocale; * Integrazione con la piattaforma esistente.

20	7	* Progettazione e implementazione di una versione semplificata della pagina per l'inserimento delle partite di Mario Kart da utilizzare in occasione del Summit <a href="#">AWS</a> di Milano.
40	8	* Stesura della documentazione di progetto delle attività di sviluppo condotte nelle settimane precedenti.
<b>Totale ore:</b>		<b>320</b>

Tabella 7.1: Consuntivo finale

## 7.2 Raggiungimento degli obiettivi

Durante il progetto di stage sono stati soddisfatti tutti i requisiti obbligatori. Il tempo pianificato per la formazione iniziale e la realizzazione del sistema di riconoscimento dei giocatori è stato adeguato.

Ha invece richiesto più tempo del previsto l'implementazione del [chatbot](#) vocale, avendo incontrato alcune difficoltà nell'interpretazione non sempre corretta da parte del bot dei nickname dei giocatori pronunciati dall'utente. Oltre a quelli classificati obbligatori, sono stati soddisfatti anche tutti i requisiti desiderabili. Sono state quindi aggiunte due pagine al sito:

- \* Una per l'inserimento dei risultati di gioco a Duck Game;
- \* Una per l'inserimento più rapido delle partite di Mario Kart attraverso riconoscimento facciale. Questa funzionalità ha permesso l'utilizzo del prodotto allo stand di Zero12 S.r.l. in occasione del Summit [AWS](#) tenutosi a Milano durante gli ultimi giorni di stage.

Per quanto riguarda invece i requisiti facoltativi, non vi è stato il tempo necessario alla loro implementazione.

Il raggiungimento degli obiettivi, rappresentato dal grado di implementazione dei requisiti, è schematizzato in [Tabella 7.2](#) e [Tabella 7.3](#).

Requisito	Stato
R0-1	Implementato
R0-1.1	Implementato
R0-1.2	Implementato

RO-1.3	Implementato
RO-1.4	Implementato
RO-1.5	Implementato
RO-1.6	Implementato
RO-2	Implementato
RO-2.1	Implementato
RO-2.2	Implementato
RD-1	Implementato
RD-1.1	Implementato
RD-1.2	Implementato
RD-1.3	Implementato
RD-2	Implementato
RF-1	Non Implementato

**Tabella 7.2:** Tracciamento dei requisiti

Requisiti	Completamento
Obbligatori	100%
Desiderabili	100%
Facoltativi	0%

**Tabella 7.3:** Completamento dei requisiti

## 7.3 Conoscenze acquisite e valutazione personale

Durante le otto settimane di stage ho avuto modo di apprendere moltissimo, sia a livello di hard-skill che di soft-skill.

Ho avuto modo non solo di studiare tecnologie che non avevo mai utilizzato (come ad esempio [AWS](#) e *Angular*), ma anche di approfondire e affinare le conoscenze che durante questi anni di università avevo acquisito.

Ho imparato inoltre a comunicare in modo più efficace non solo le mie idee ma anche le mie difficoltà, in modo che altri potessero aiutarmi e consigliarmi. Mi ha aiutato anche a comprendere meglio il funzionamento corretto e proficuo di un rapporto collaborativo tra colleghi con lo scopo di raggiungere un obiettivo comune.

Questo progetto di stage mi ha permesso di sperimentare, mettermi alla prova e utilizzare la mia creatività per realizzare un prodotto finale che incontrasse le aspettative dell'azienda.

Valuto dunque molto positivamente l'esperienza di stage svolta presso Zero12 S.r.l. poiché mi ha permesso di entrare in contatto con un ambiente lavorativo reale, di mettere a frutto le mie conoscenze pregresse e svilupparne di nuove.



# Glossario

**API** in informatica con il termine *Application Programming Interface API* (ing. interfaccia di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. [8](#)

**Automatic Speech Recognition** campo dell'informatica che sviluppa metodologie e tecnologie che permettano il riconoscimento e la traduzione di linguaggio parlato in testo in modo automatico attraverso l'utilizzo del computer. [3](#)

**AWS** azienda di proprietà Amazon che fornisce servizi di cloud computig, disponibili sull'omonima piattaforma. [1](#), [7](#), [8](#), [11](#), [13](#), [14](#), [17](#), [19](#), [28](#), [36](#), [37](#)

**chatbot** software che simula conversazioni simili a quelle con esseri umani attraverso chat scritte o vocali. Il suo compito è quello di aiutare gli utenti a svolgere specifiche attività rispondendo nel modo corretto alle richieste. [4](#), [5](#), [9](#), [27-33](#), [36](#)

**CloudFront** Amazon CloudFront è un servizio di rete di distribuzione di contenuti (CDN). Esso fornisce una rete globalmente distribuita di server che mantengono i contenuti (come video o media) in modo da aumentare la velocità di accesso a tali contenuti quando vengono richiesti. [17](#)

**controllo di versione** sistema che registra nel tempo i cambiamenti di uno o più file, così da poter richiamare una specifica versione in un secondo momento. [11](#)

**CORS** (*Cross-origin resource sharing*) meccanismo che permette di definire restrizioni sulle risorse che possono essere richieste da domini diversi da quello su cui tali risorse sono allocate. [14](#), [15](#)

**CSS** (*Cascading Style Sheets*) è un linguaggio di stile utilizzato per definire la formattazione di documenti [HTML](#), come ad esempio pagine web. [10](#)

**deploy** procedura di rilascio di un sistema software o di un'applicazione. [13](#), [15](#), [17](#)

**distanza di Levenshtein** misura per la differenza fra due stringhe. In particolare, la distanza di Levenshtein tra due stringhe A e B è il numero minimo di modifiche elementari che consentono di trasformare la A nella B. Per modifica elementare si intende la cancellazione di un carattere, la sostituzione di un carattere con un altro o l'inserimento di un carattere.. [30](#)

**Elo** prova. [16](#)

**FaaS** categoria dello sviluppo cloud che fornisce una piattaforma che permetta agli sviluppatori di eseguire e gestire le varie funzionalità della propria applicazione senza necessità di gestire e mantenere infrastrutture fisiche. *FaaS* è una delle modalità con cui si può costruire un'architettura [serverless](#). [7](#)

**framework** architettura logica riutilizzabile sulla quale un software può essere progettato. È definito da un insieme di classi astratte e dalle relazioni tra di esse e può includere programmi, librerie e strumenti di supporto. [7](#), [10](#), [13](#)

**HTML** (*HyperText Markup Language*) è un linguaggio di markup che definisce le modalità di impaginazione del contenuto di pagine web attraverso l'utilizzo di *tag*. [10](#), [39](#)

**Natural Language Understanding** campo dell'intelligenza artificiale che si occupa della comprensione del testo da parte delle macchine. [3](#)

**NoSQL** tipologia di database che fornisce un meccanismo per salvare dati diverso dal concetto di *tabella*, tipico dei database relazionali. Essi salvano i dati utilizzando strutture differenti, come ad esempio oggetti di tipo *chiave-valore* o *documento*. [8](#)

**serverless** modello di esecuzione cloud che non richiede la presenza di server fisici che devono essere mantenuti e configurati. I server sono comunque presenti ma si trovano nel cloud e allocano risorse dinamicamente appena queste vengono richieste. Quando l'applicazione non è in uso, nessuna risorsa viene consumata ed il prezzo risulta quindi essere pari a zero. [7](#), [8](#), [13](#), [40](#)

**skill** equivalente di un'applicazione, sviluppata per l'assistente intelligente Alexa. [5](#), [6](#)

**YAML** con l'acronimo ricorsivo (*YAML Ain't a Markup Language*) formato per la serializzazione di dati leggibile da esseri umani. Questo formato può essere utilizzato come file di configurazione, come nel caso del *Serverless Framework*. [7](#), [13](#)

# Bibliografia

## Siti web consultati

*Amazon Web Services*. URL: <https://aws.amazon.com/it/>.

*Documentazione Amazon Lex*. URL: <https://docs.aws.amazon.com/lexv2/latest/dg/what-is.html>.

*Documentazione Amazon Rekognition*. URL: <https://docs.aws.amazon.com/rekognition/latest/dg/what-is.html>.

*Documentazione Amazon S3*. URL: <https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html>.

*Documentazione Angular*. URL: <https://angular.io/docs>.

*Documentazione AWS DynamoDB*. URL: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html>.

*Documentazione AWS Lambda*. URL: <https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>.

*Documentazione AWS SDK per JavaScript*. URL: <https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/>.

*Documentazione componenti Nebular*. URL: <https://akveo.github.io/nebular/docs/components/components-overview>.

*Documentazione Serverless framework*. URL: <https://www.serverless.com/framework/docs>.