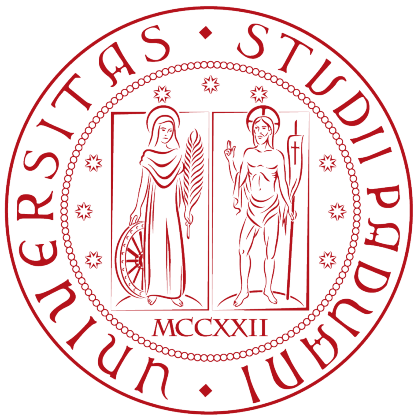
Università degli Studi di Padova

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 trecento 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. NomeDelProfessore, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

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

Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme e le mille avventure vissute.

Padova, Luglio 2022 Nicla Faccioli

Indice

Elenco delle figure

Elenco delle tabelle

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 va 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 approfondisce lo sviluppo del sistema di image recognition;

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* pe riconoscere i giocatori e 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. Nel corso dello stage il livello di obbligatorietà di tali requisiti è variato in risposta alle esigenze dell'azienda.
Di seguito viene riportata la versione finale dell'analisi effettuata.

2.3.1 Requisiti obbligatori

Di seguito vengono elencati i requisiti obbligatori:

* Sviluppo di un micro-servizio per le attività di *face detection* e *recognition*;
* Sviluppo di un micro-servizio per le attività di controllo vocale via web

2.3.2 Requisiti desiderabili

Di seguito vengono elencati i requisiti desiderabili:

* Integrazione di un nuovo gioco all'interno della piattaforma;
* Implementazione di una versione semplificata per l'inserimento dei dati di Mario Kart per l'utilizzo della piattaforma in occasione del Summit AWS di Milano.

2.3.3 Requisiti facoltativi

Di seguito vengono elencati i requisiti facoltativi:

* Sviluppo di una skill Alexa con le stesse funzionalità del chatbot vocale richiesto come requisito obbligatorio e integrato sulla piattaforma web.

2.4 Pianificazione

La durata complessiva dello stage è stata di 8 settimane di lavoro a tempo pieno per un totale di circa 320 ore.

Secondo il piano di lavoro iniziale definito con l'azienda, le attività sono distribuite come segue:

Durata in ore	Settimana	Descrizione
40	1	* Studio delle tecnologie necessarie.
80	2, 3	* Progettazione e sviluppo di un micro-servizio per attività di *face detection* 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	∗ Sviluppo della skill Alexa per il controllo vocale e l'aggiornamento dei risultati; ∗ Integrazione con la piattaforma esistente.
40	8	∗ Testing e stesura della documentazione di progetto delle attività di sviluppo condotte nelle settimane precedenti.
Totale ore:		**320**

Tabella 2.1: Pianificazione delle attività

Durante il periodo di stage in azienda il piano di lavoro ha subito modifiche e di conseguenza la versione finale della pianificazione riportata nel Capitolo 7 diverge da quella qui presentata. Tali modifiche sono state effettuate in risposta alle esigenze e richieste dell'azienda. Durante tutta la durata del tirocinio sono stati effettuati stand-up giornalieri con il tutor aziendale in affiancamento per monitorare lo stato di avanzamento ed evidenziare eventuali problemi sorti.

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. Per effettuare il deploy delle Lambda, delle tabelle DynamoDB e dei bucket S3 deifiniti sarà sufficiente eseguire il comando `serverless deploy`.

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 codsce 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 Application Program Interface 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 documento o 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 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'intero 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.

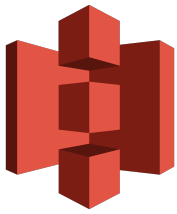


Figura 3.6: Logo Amazon S3

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 facciale per il riconoscimento dei giocatore in fase di inizializzazione di una partita.

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 per permettere lo sviluppo di interfacce di comunicazione all'interno di applicazioni software. Nel progetto è stato utilizzato per implementare un chatbot vocale per 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 manutenuto 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 associando 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 surante 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 politica aziendale.

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 permette il 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*, un framework che permette di costruire architetture serverless. Esso consente inoltre di definire l'architettura AWS 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 le 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*. Specificando
* **Http** permette di definire degli *API Gateway* HTTP endpoint che quando chiamati provocano l'esecuzione della funzione;
* **path** definisce il path 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 presenti già 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 ad uno dei tre giochi disponibili (Mario Kart, Calcetto e Duck Game). Questi risultati vengono memorizzati all'interno di tabelle DynamoDB dedicate e vengono utilizzati per aggiornare il punteggio Elo dei giocatori che hanno partecipato.

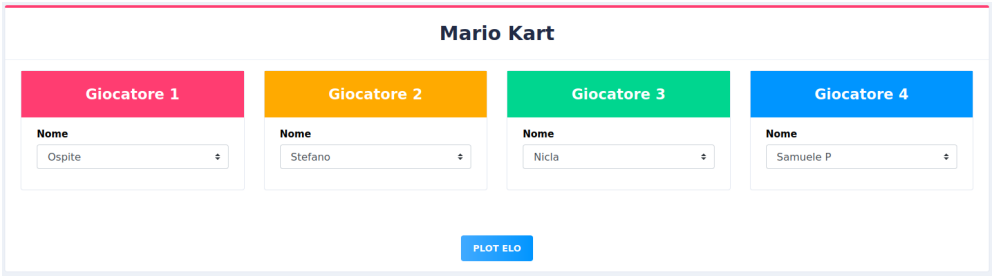


Figura 4.4: Form per l'inserimento di una nuova partita

Predizione dei risultati di nuove partite

Nell'applicativo è presente un algoritmo di *Machine Learning* che, utilizzando 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.

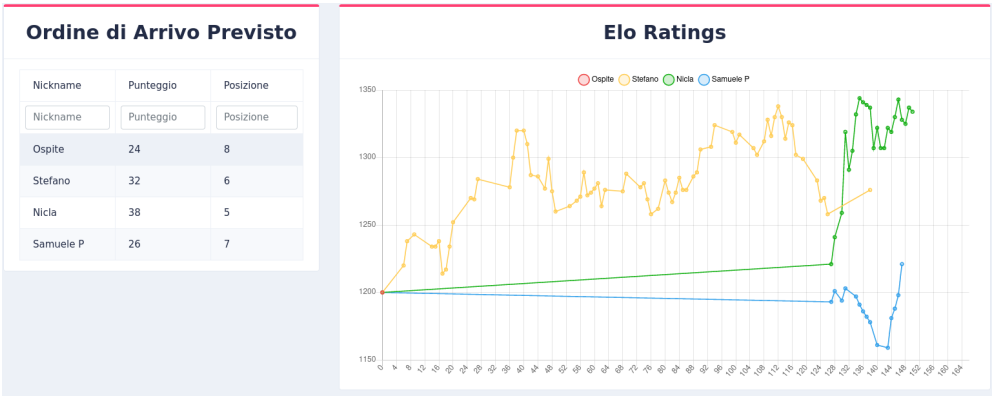


Figura 4.5: Visualizzazione della predizione di una nuova partita

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, sempre all'interno della stessa pagina viene mostrato un grafico con lo storico di tutti i punteggi per i dieci giocatori più frequenti per lo 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 permetterà di visualizzare le modifiche effettuate.

Capitolo 5

Integrazione di Amazon Rekognition

In questo capitolo viene approfondito lo sviluppo e l'integrazione del sistema di image recognition

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 servizio di riconoscimento facciale implementato.
Inoltre viene presentato il design dell'applicazione web dopo l'integrazione.

5.2.1 Architettura

Il micro-servizio sviluppato fa uso di diversi servizi messi a disposizione da AWS:

* *Amazon Rekognition*;
* *Amazon S3*;
* *AWS Lambda*

Nei prossimi paragrafi vengono descritti nel dettaglio tali servizi, il loro ruolo all'interno dell'applicativo e come essi 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 le funzioni utilizzate:

* `DetectFaces`: individua le cento facce di dimensione maggiore presenti nell'immagine. Per ogni viso individuato ne restituisce i dettagli, 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 ne salva solamente le 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 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 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 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 specificato è *image/jpeg*

* **"putObject"**: tipo di operazione che si vuole effettuare con l'URL generato. In questo caso si tratta di una 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 lasciare all'interno del *bucket* dell'inutile spazzzatura, è stata impostata una *lifecycle rule* in modo tale che tutte le immagini caricate (con prefisso *img/*) vengano eliminate in modo automatico dopo un giorno dal loro caricamento.

5.2.1.3 AWS Lamda

Di seguito sono descritte le funzioni *Lambda* implementate per la funzionalità di riconoscimento facciale:

* `getPresignedUpload`: richiede il presigned URL per il caricamento dell'immagine;

* `callRekognition`: effettua il *download* dell'immagine dal *bucket* S3;

* `elaborateImage`: effettua le chiamate effettive a *Rekognition*. In particolare:

    1. Individua e ritaglia i volti nella fotografia utilizzando `rekognition.IndexFace`;
    2. Per ciascuna delle facce individuate, chiama `rekognition.SerachFacesByImage` per cercare corrispondenze all'interno della *collection*;
    3. Se sono state trovate delle corrispondenze restituisce il *faceId* più probabile;
    4. Altrimenti restituisce il crop del viso non riconosciuto.

* `updatePlayer`: aggiorna un giocatore già presente nel database con il nuovo *faceId* associatogli;

* `indexFace`: utilizza `rekognition.IndexFaces` per indicizzare nuove facce all'interno della *collection* di *Rekognition*.

5.2.2 Funzionamento generale

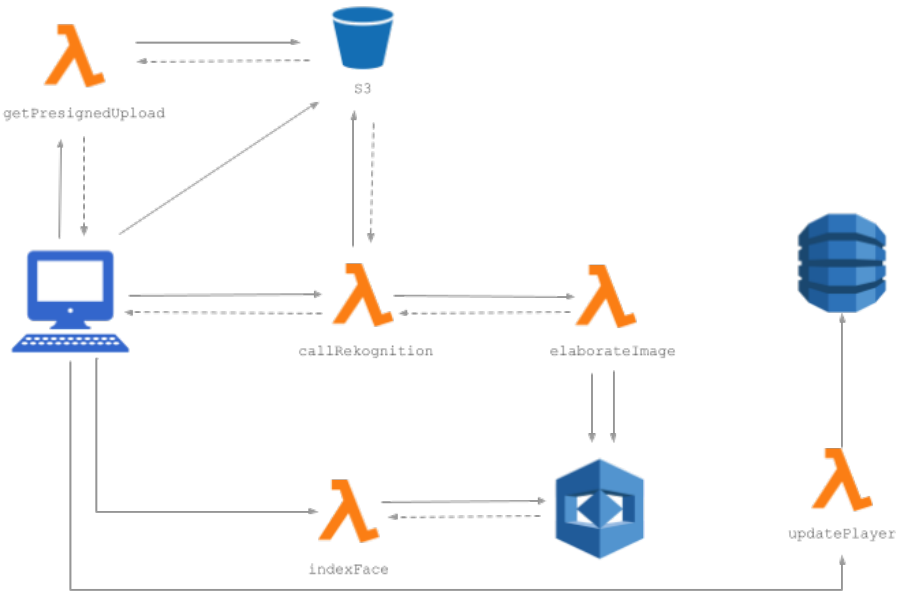


Figura 5.2: Grafico del funzionamento del servizio di image recognition

Segue la descrizione della Figura 5.2:

* Il front-end utilizza la funzione `getPresignedUpload` per ottenere l'URL per il caricamento dell'immagine nel *bucket* di *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*;
* `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;
  - Il nuovo utente viene registrato nel database con 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, la realizzazione dell'interfaccia è avvenuta utilizzando *Angular 12.x* in combinazione con la libreria *Nebular*, specifica per lo sviluppo 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 il riconoscimento.

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**: 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 non fossero presenti all'interno della fotografia scattata, è possibile inserirli manualmente. Una volta inserito il nickname desiderato

viene mostrata la card corrispondente.
Il numero di giocatori selezionabili per ciascuna squadra è esattamente due.

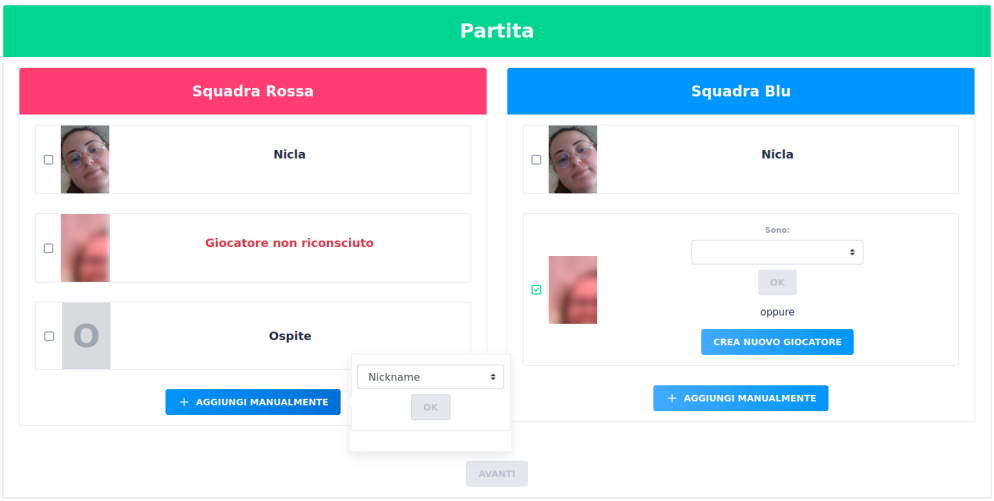


Figura 5.4: Selezione dei giocatori

Dopo la selezione dei giocatori sarà possibile scambiare i due nickname all'interno di ciascuna squadra per associare loro il ruolo desiderato e formare le squadre definitive (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*.

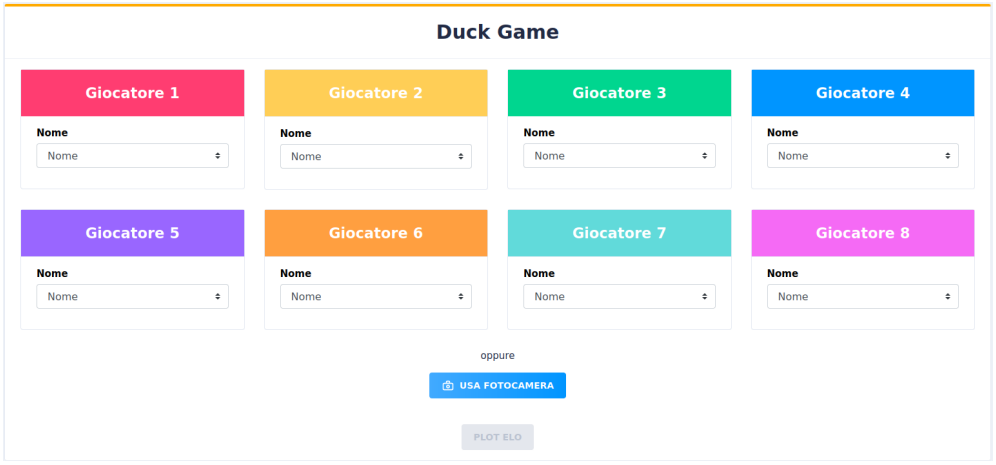


Figura 5.6: Schermata iniziale di Duck Game

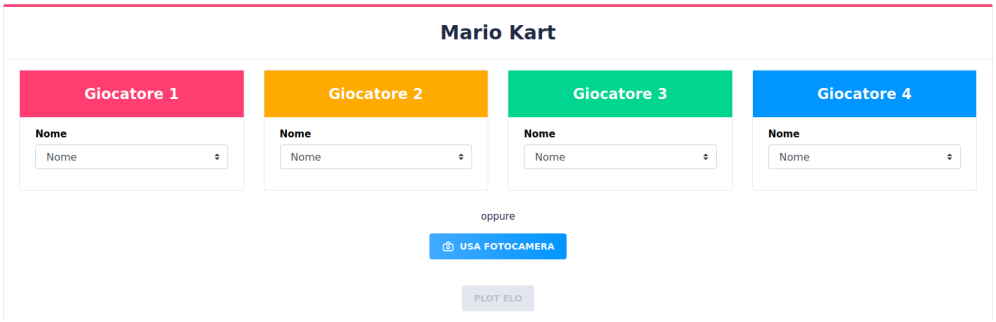


Figura 5.7: Schermata iniziale di Mario Kart

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 non fossero presenti all'interno della fotografia scattata, è possibile inserirli manualmente. Una volta inserito il nickname desiderato viene mostrata la card corrispondente.

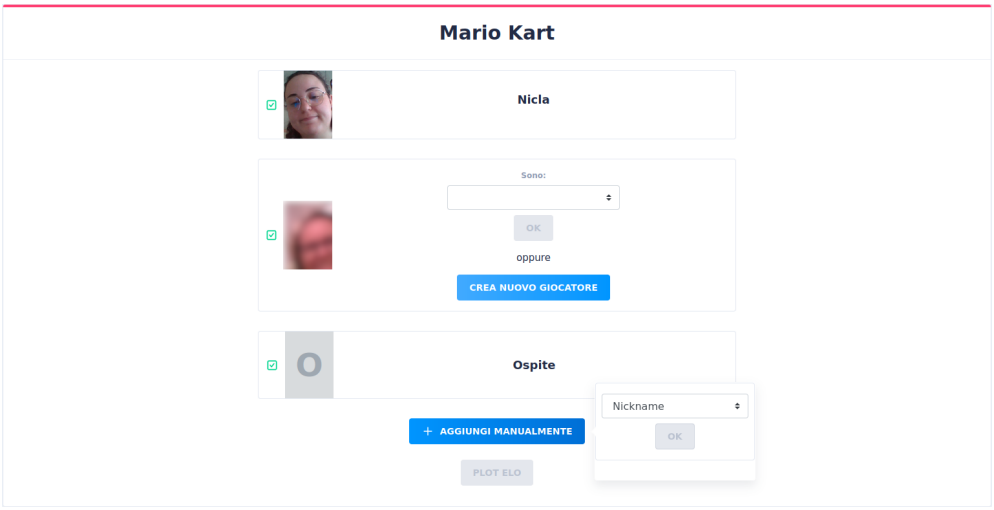


Figura 5.8: Selezione dei giocatori di Mario Kart

Capitolo 6

Integrazione di Amazon Lex

In questo capitolo viene approfondito lo sviluppo e l'integrazione del sistema di interazione vocale

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 caricamento di una nuova partita 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;
  - Gol effettuati da ciascun ogni giocatore;
* **Mario Kart**:
  - Torneo giocato;
  - Posizione finale di ciascun giocatore;
  - Punteggio finale di ciascun giocatore.
* **Duck Game**:
  - Punteggio di ciascun giocatore.

Poiché l'inserimento di questi dati risultava laborioso, è stato richiesto di implementare un chatbot vocale che permettesse di inizializzare e salvare le 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*;
* *Amazon S3*;
* *AWS Lambda*.

6.2.1.1 Amazon Lex

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 verisone 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:

* **Frasi d'esempio**: frasi che l'utente deve comunicare 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 *Lambda* da richiamare alla fine di un *intent* oppure ogni volta che viene fornito un valore per uno *slot*.
Le funzioni di *Lex* utilizzate sono le seguenti:

* `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

presigned upload e download, bucket per lo scambio degli audio

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 specificato è *audio/wav*
* **"putObject"**: tipo di operazione che si vuole effettuare con l'URL generato. In questo caso si tratta di una 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 lasciare all'interno del *bucket* dell'inutile spazzzatura, è stata impostata una *lifecycle rule* in modo tale che tutti gli audio caricati (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.

* `getPresignedDownloadAudio`: richiede il presigned 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.

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 nei nominativi, verrà richiamato automaticamente il secondo *intent* per permettere l'inserimento dei risultati.

Il funzionamento generale dell'applicazione è il seguente:

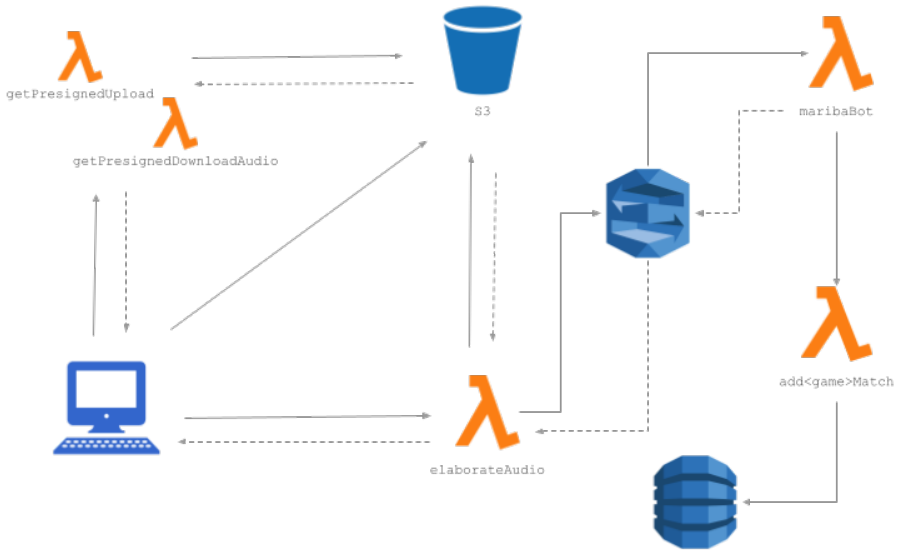


Figura 6.2: Grafico del funzionamento del chatbot

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 `elaborateImage` 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 contenete la risposta generata dal chatbot;
7. `elaborateAudio` carica la risposta su *S3* e restituisce le informazioni della posizione dell'oggetto nel 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 combinazione 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: Stato di registrazione del pulsante

Elaborazione dell'audio

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

Riproduzione dell'audio ricevuto da Lex

L'applicativo sta riproducendo l'audio ricevuto in risposta dal chatbot.

Pronto per la registrazione della risposta

L'applicativo è pronto per la registrazione da parte dell'utente della prossima interazione con il chatbot.

Stato di errore

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

Capitolo 7

Conclusioni

7.1 Consuntivo finale

Durata in ore	Settimana	Descrizione
40	1	∗ Studio delle tecnologie necessarie.
100	2, 3, 4	∗ Progettazione e sviluppo di un micro-servizio per attività di *face detection* per la creazione di squadre di gioco; ∗ Integrazione con la piattaforma esistente.
40	4, 5	∗ 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.

40	8	∗ Stesura della documentazione di progetto delle attività di sviluppo condotte nelle settimane precedenti.
Totale ore:		**320**

Tabella 7.1: Pianificazione delle attività

7.2 Raggiungimento degli obiettivi

7.3 Conoscenze acquisite

7.4 Valutazione personale

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 trduzione 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, 4, 7, 8, 11, 13, 14, 17, 19, 28

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, 9, 27–33

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 contentuti 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 seconodo momento. 11

CORS *(Cross-origin resource sharing)* meccanismo che permette di defnire restrizioni sulle risorse che possono essere richieste da domani 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. 7, 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

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, 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, 37

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 comunuque presetni 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, 13, 37

skill equivalente di un'applicazione, sviluppata per l'assistente intelligente Alexa. 4

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.