



MOCC

Meal Optimizer Cloud Chef

Mario Cosenza



INDICE PRESENTAZIONE

01

INTRODUZIONE

Qual è lo scopo di MOCC

02

ARCHITETTURA

Illustrazione dei servizi Azure usati con il relativo costo

03

API E FRONTEND

Descrizione dell'api GraphQL delle app in Flutter e Python

04

CONCLUSIONI

Analisi delle limitazioni e proposte di sviluppo

01.

INTRODUZIONE



Qual è lo scopo di MOCC



Introduzione



MOCC è l'assistente di **cucina intelligente** che rivoluziona la gestione domestica, trasformando la dispensa in un hub digitale connesso.

Ottimizza la spesa e **riduce gli sprechi alimentari** grazie all'integrazione di intelligenza artificiale per generare **ricette su misura** con ciò che possiedi.





Gestione Intelligente e Sociale

Inventario Condiviso

Scansiona scontrini via **OCR** per popolare automaticamente la dispensa e coordinati con la famiglia tramite frigoriferi digitali sincronizzati in tempo reale.

Chef AI Generativo

Non sai cosa cucinare? L'AI analizza i tuoi ingredienti, **scadenze imminenti** e preferenze dietetiche per creare ricette uniche passo dopo passo.

Community & Sicurezza

Condividi le tue creazioni culinarie in un **feed social sicuro**, dove i contenuti sono moderati automaticamente per garantire un ambiente positivo e ispirazionale.

Riduzione Sprechi

Ricevi notifiche proattive sui prodotti in scadenza, trasformando **potenziali rifiuti** in opportunità gastronomiche.



02.

ARCHITETTURA

Servizi di Azure usati con il relativo costo e RBAC

Distribuzione dei Servizi

AI
Hub intelligenza
gestito



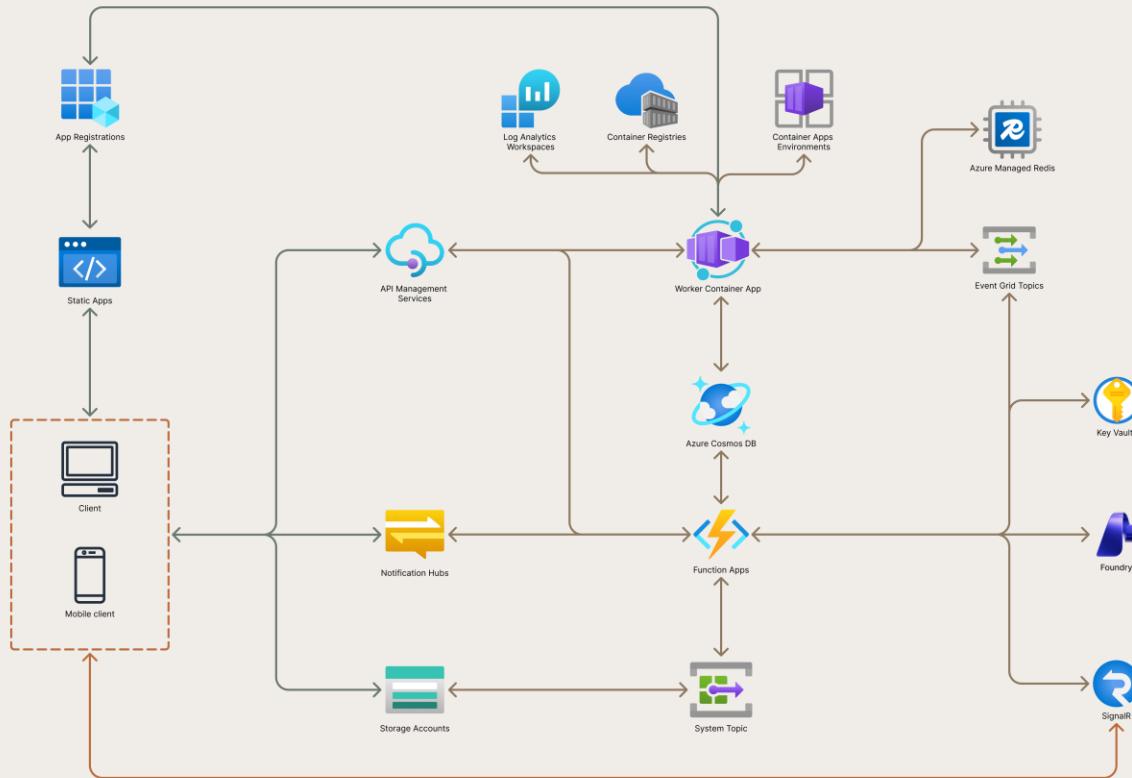
Integrazione
Orchestrazione
eventi sicura



Computazione
Hosting elastico
performante



Schema Architettura





Azure Container Apps

```
resource app 'Microsoft.App/containerApps@2025-07-01' = {  
    name: webAppName  
    properties: {  
        managedEnvironmentId: caEnv.id  
        configuration: {  
            ingress: { external: true, targetPort: 8080 }  
            registries: [{ server: acr.properties.loginServer, username: acr.name }]  
        }  
        template: {  
            containers: [{  
                image: selectedImage  
                resources: { cpu: json('0.75'), memory: '1.56i' }  
            }]  
            scale: { minReplicas: 0, maxReplicas: 10 } // Scaling gestito da KEDA  
        }  
    }  
}
```



Compute

Risorse ottimizzate con **0.75 vCPU e 1.5Gi di Memoria.**

Log Analytics

Monitoraggio centralizzato tramite **Log Analytics Workspace** per debugging e osservabilità.

Container Registry (ACR)

Gestione privata e sicura delle immagini **Docker** su Azure.

KEDA Scaling

Auto-scaling dinamico (es. su traffico HTTP) con supporto **scale-to-zero**.



ACA VS APP SERVICES



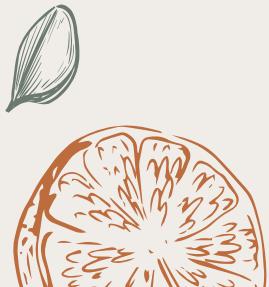
ACA*

Serverless, paga al consumo,
scala a zero. Ottimizzato per
microservizi, container e
architetture moderne.



APP SERVICES

Hosting PaaS tradizionale per
web app e monoliti. Costi fissi e
scaling basato su risorse.



Managed Redis e Cosmos DB



Managed Redis

Cache Enterprise (Balanced B0) per alte prestazioni. Gestione sessioni e caching.



Cosmos DB

Database NoSQL con ridondanza di zona attiva. Scalabilità autoscale impostata a 400 RU/s massimi.

Contenitori CosmosDB



Inventory

Gestione dello stato dei prodotti, quantità e scadenze.



Users

Storage dei profili utente e punteggi accumulati.



Cookbook

Archivio delle ricette generate dall'IA o inserite manualmente.



History

Registro cronologico degli acquisti effettuati.



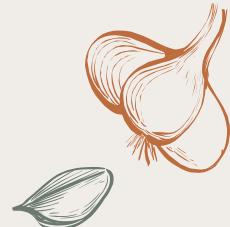
Social

Gestione dei post fotografici e dei relativi commenti.



Leaderboard

Dati aggregati per le classifiche e la competizione tra utenti.





Azure Functions App

```
- □ ×  
  
resource plan 'Microsoft.Web/serverfarms@2025-03-01' = {  
  name: planName  
  location: location  
  kind: 'functionapp'  
  sku: { name: 'FC1', tier: 'FlexConsumption' }  
}  
  
// Configurazione Function App Python 3.12  
resource functionApp 'Microsoft.Web/sites@2025-03-01' = {  
  name: functionName  
  kind: 'functionapp,linux'  
  identity: { type: 'SystemAssigned' }  
  properties: {  
    serverFarmId: plan.id  
    siteConfig: {  
      linuxFxVersion: 'PYTHON|3.12'  
      appSettings: [  
        { name: 'APPLICATIONINSIGHTS_CONNECTION_STRING',  
          value: appInsights.properties.ConnectionString  
        }, { name: 'COSMOS_URL', value: cosmosDbEndpoint }  
      ]  
    }  
  }  
}
```



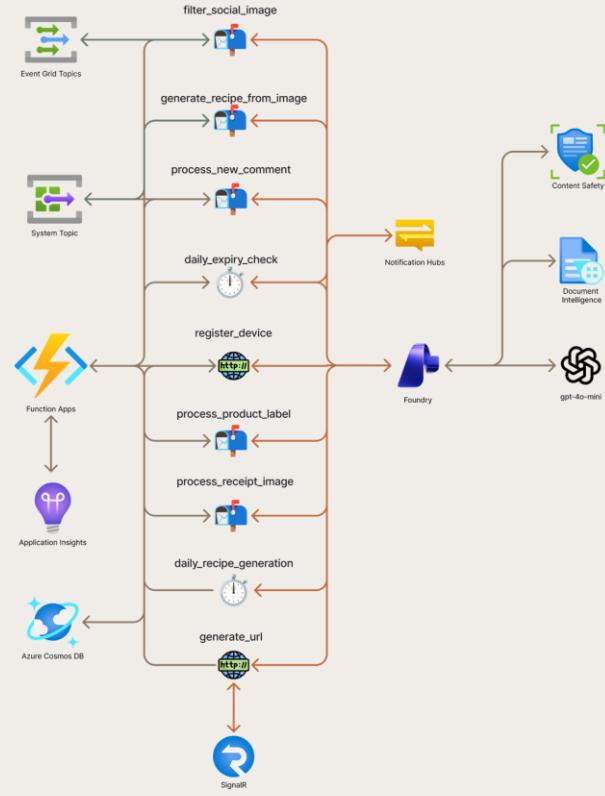
Le Azure Functions gestiscono tutti i processi **asincroni e reazionari** del sistema, sgravando il backend principale (ACA) da task intensivi:

- **AI Processing:** Generazione ricette tramite integration con Azure OpenAI.
- **Blob Triggers:** Elaborazione automatica degli scontrini caricati su Storage.
- **Scheduled Tasks:** Pulizia automatica e notifiche di scadenza prodotti via Notification Hub.



Il sistema adotta il runtime Python 3.12, garantendo la massima sicurezza grazie all'accesso *keyless* verso Key Vault e Cosmos DB tramite Managed Identity e un networking protetto da restrizioni IP e Service Tags dedicati per Event Grid.

Schema Azure Functions



Confronto Piani Functions App

	Consumption	Flex Consumption*
Piano (SKU)	Y1 (In dismissione)	FC1
Networking	Solo IP pubblici	VNET e restrizioni CIDR
Latenza	Cold start elevato	Supporto per Pre-warming
Deployment	Basato su file zip	App-package (via Azure Blob Storage)

*Piano usato in MOCC

EVENT GRID

Event Grid gestisce il disaccoppiamento tra il backend, lo storage e le funzioni Python.
Utilizza sia **System Topics** che **Custom Topics** (`CloudEventSchemaV1_0`).

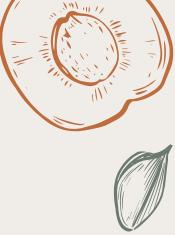
	Topic	Sottoscrizione	Sorgente	Destinazione
System	moccblob eventgrid	receipt-processed	<i>BlobCreated</i>	<code>process_receipt_image</code>
System	moccblob eventgrid	image-processed	<i>BlobCreated</i>	<code>generate_recipe_from_image</code>
System	moccblob eventgrid	social-posts-sub	<i>BlobCreated</i>	<code>filter_social_image</code>
System	moccblob eventgrid	label-processed	<i>BlobCreated</i>	<code>process_product_label</code>
Custom	moccpost comments	newComment	<i>Social. CommentAdded</i>	<code>process_new_comm</code>

Azure API Management

APIM centralizza il controllo dell'intero ecosistema MOCC, agendo come **scudo di sicurezza e punto di accesso unico** per il frontend Flutter.

Semplifica lo sviluppo garantendo che ogni richiesta sia pre-validata e correttamente instradata senza esporre i servizi interni.

- **GraphQL Facade:** Espone e valida lo schema applicativo, ottimizzando la comunicazione tra Client e Server.
- **Sicurezza Centralizzata:** verifica dei token **JWT**, gestendo identità ed autorizzazioni a monte.
- **Orchestrazione Traffico:** Smista intelligentemente le richieste tra backend Go e logiche Python (es. negoziazione SignalR).
- **Gestione CORS:** Abilita nativamente il supporto per l'app Web ed i vari client, eliminando colli di bottiglia infrastrutturali.



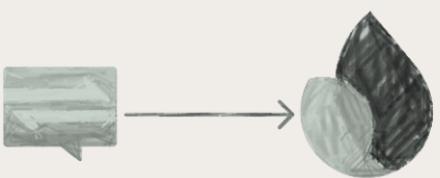
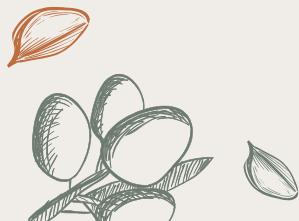
Notification Hub e SignalR

Notification Hub

- **Firebase FCM V1 Integration:** Configurato con credenziali native (Client Email, Private Key e Project ID) per supportare il nuovo standard di Google.
- **Multi-Platform:** Astrae la complessità di invio verso Android/iOS, gestendo le registrazioni dei dispositivi tramite tag basati sull'ID utente.
- **Integrazione Bicep:** I parametri @secure() garantiscono la protezione delle chiavi private Firebase durante il deployment.

SignalR

- **Serverless Mode:** Ideale per l'architettura a microservizi, eliminando la necessità di gestire connessioni persistenti sul backend Go o Python.
- **Instant Sync:** Utilizzato per aggiornare in tempo reale la lista della spesa e lo stato dei frigoriferi condivisi tra più utenti.
- **CORS & Autenticazione:** Accesso protetto e regolato da APIM per garantire comunicazioni sicure e cross-platform.



Azure Foundry

Document Intelligence

Analisi OCR avanzata



GPT 4o-mini

Reasoning AI efficiente



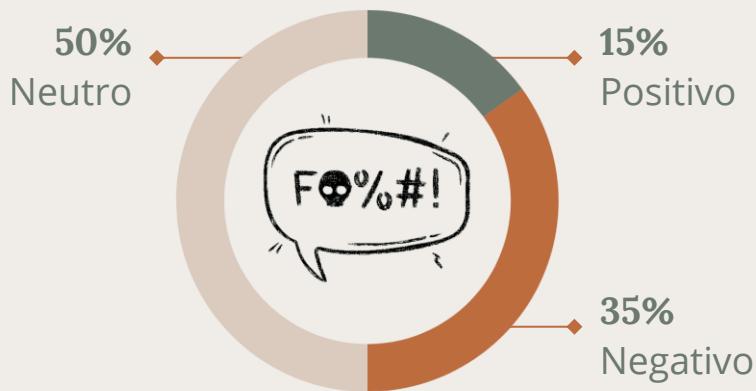
Content Safety

Filtro contenuti nocivi



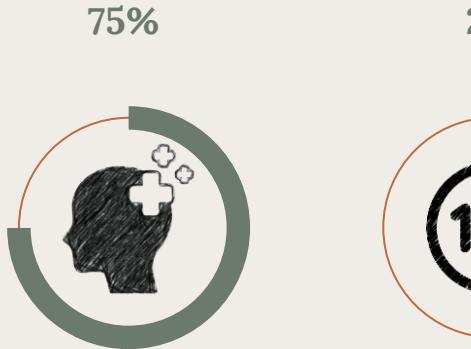
Content Safety

Hate/Violence



Content Safety protegge l'ecosistema social di MOCC **analizzando testo ed immagini** per rilevare **contenuti inappropriati** prima che vengano visualizzati dalla **community**.

Valutazione del contenuto



Self-Harm

Riferimenti ad autolesionismo

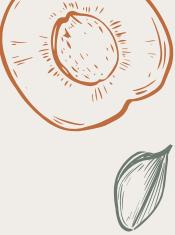


Sexual Content

Contenuti esplicativi o allusivi

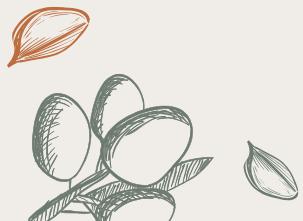
Pipeline Autenticazione



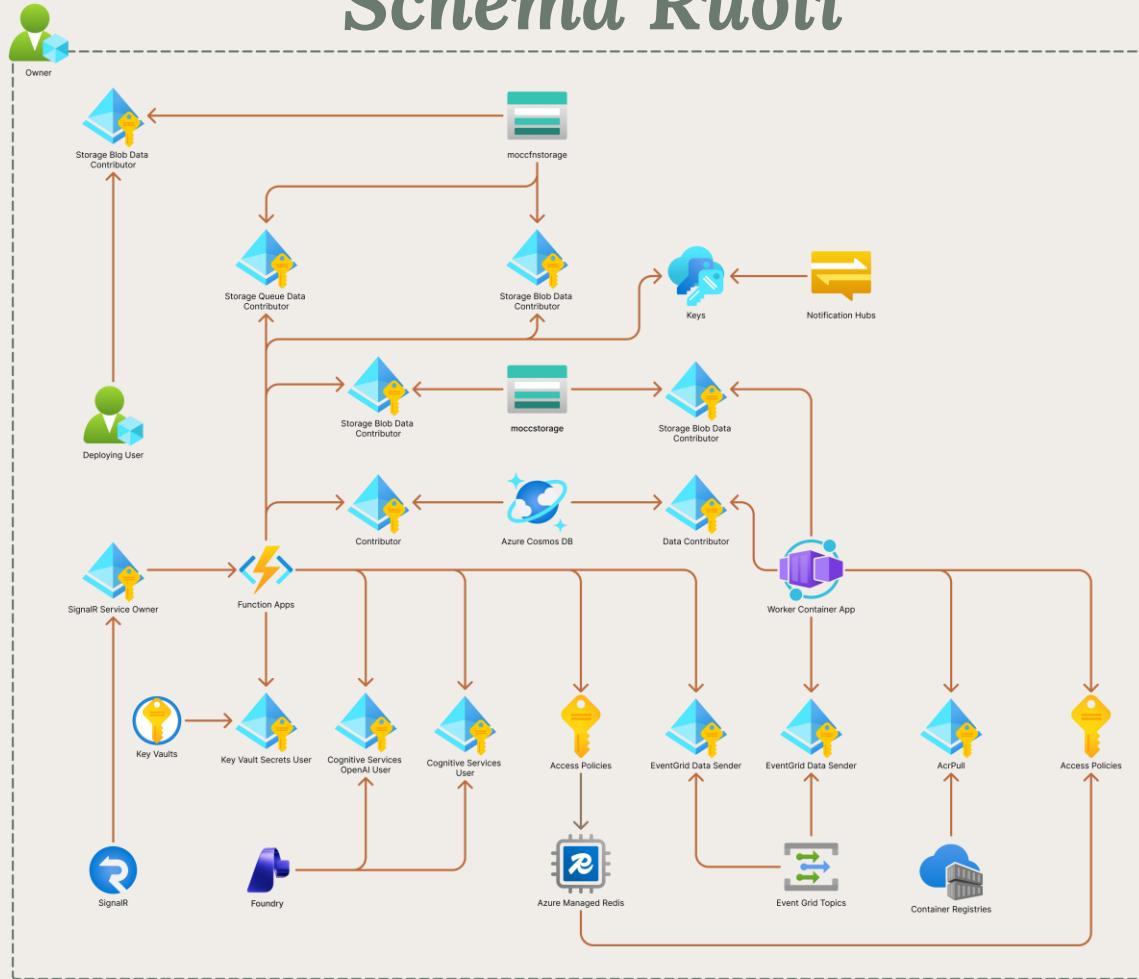


Altri Servizi

- **Azure Static Web Apps:** Hosting globale per il frontend web con integrazione CI/CD nativa.
- **Azure Key Vault:** Gestione sicura e centralizzata di segreti, chiavi e certificati.
- **Azure Container Registry (ACR):** Registro privato per l'archiviazione e gestione delle immagini Docker.
- **Azure Storage Accounts:** Persistenza per blob (scontrini, ricette) e file di sistema delle Functions.
- **Log Analytics Workspace:** Repository centrale per la telemetria e l'analisi dei log di tutta l'infrastruttura.
- **Application Insights:** Monitoraggio APM per il tracciamento delle performance e debugging proattivo.
- **Azure AI Foundry:** Piattaforma unificata per l'integrazione, il test e il monitoraggio dei modelli AI.
- **Azure Policy:** Governance automatizzata (es. limite 400 RU/s su Cosmos DB) per il controllo dei costi.
- **Container Apps Environment:** Infrastruttura isolata che gestisce rete, certificati e scaling per i microservizi.
- **RBAC (Role-Based Access Control):** Sistema di permessi granulari per la comunicazione sicura tra i vari servizi Azure.



Schema Ruoli



Analisi dei Costi





< €20 / mese



Costo di esercizio dell'intera architettura MOCC



€0,55 / giorno

Costo operativo medio dell'intera piattaforma MOCC

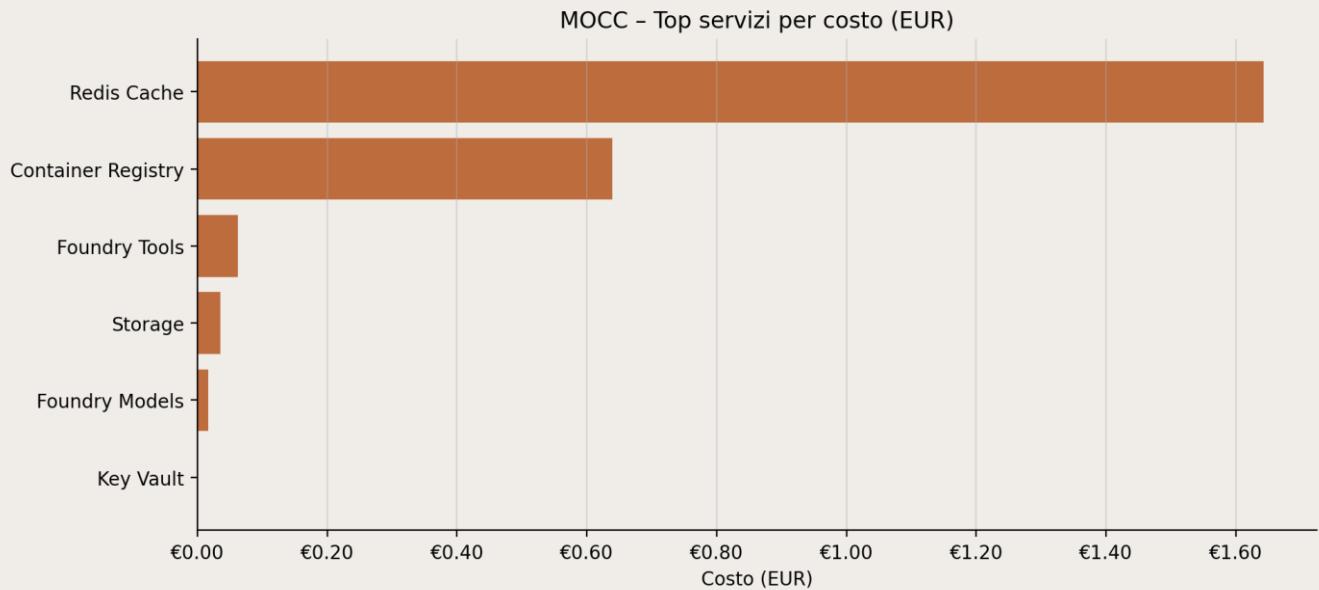
≈ 98%

Spesa concentrata su 2 servizi

< €1

Picco massimo giornaliero

Costi per servizio



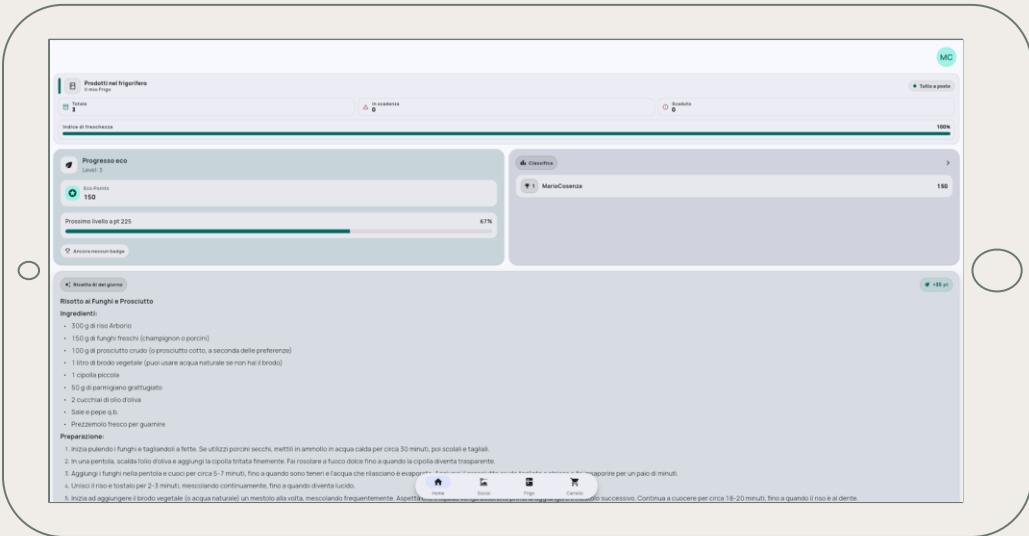
Costi osservati dal 26 gennaio al 5 febbraio

03.

API E FRONTEND

Descrizione delle applicazioni Frontend e Backend

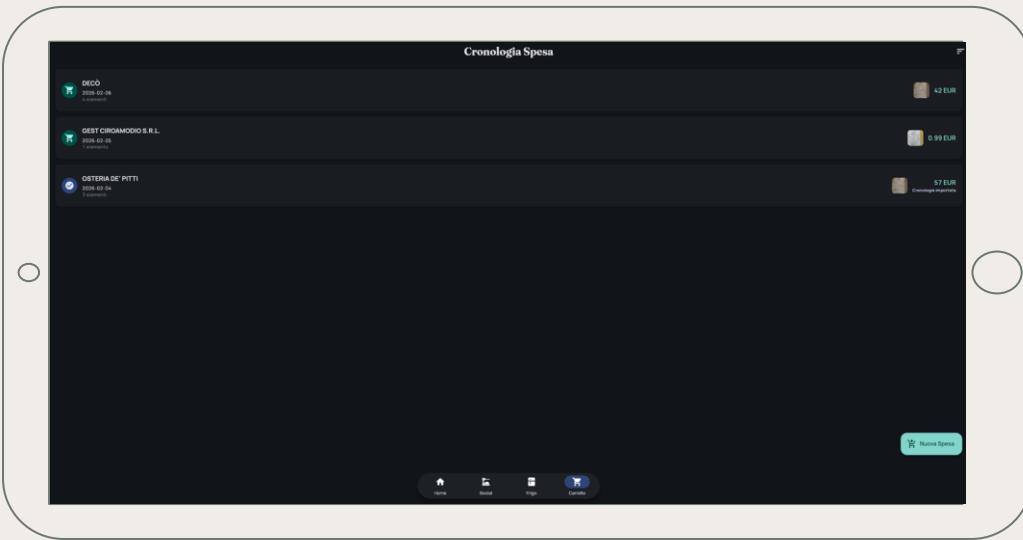
Applicazione Flutter



L'interfaccia di MOCC è progettata seguendo i principi del **Material Design 3**, puntando su un'estetica moderna, pulita e altamente reattiva.

L'esperienza utente è arricchita da **interazioni fluide** e un sistema di navigazione intuitivo, pensato per facilitare la gestione del frigorifero e la scoperta di nuove ricette.

Flutter - Librerie Core



- **go_router:** Navigazione dichiarativa e gestione fluida dei percorsi tra le viste.
- **flutter_riverpod:** Gestione dello stato reattiva per dati sempre sincronizzati.
- **graphql_flutter:** Comunicazione ottimizzata e caching efficiente con il backend.

Gestione autenticazione in Flutter



MSAL Integration

Integrazione nativa con **Microsoft Authentication Library** per il login sicuro tramite Entra ID.



Token Management

Acquisizione e refresh automatico dell'**Access Token** per garantire sessioni utente persistenti.



Secure Storage

Archiviazione crittografata delle credenziali e **iniezione automatica** dei token negli header GraphQL.





API GraphQL con GO

Il **backend** di MOCC utilizza **Go** per garantire risposte rapide e un'impronta di memoria minima. Il servizio viene distribuito come immagine **Docker**, assicurando un ambiente di runtime isolato e coerente su Azure.

- **Gqlgen:** Utilizzo della libreria leader [99designs/gqlgen](#) per la generazione automatica dei tipi e la gestione dei resolver.
- **Monitoraggio:** Implementazione dell'endpoint /health per il controllo dello stato vitale da parte dell'orchestratore di Azure.

```
// Resolver autogenerato: r rappresenta la struct che coordina i dati
func (r *queryResolver) Fridge(ctx context.Context, id string) (*model.Fridge, error)
{    // r.Logic contiene l'integrazione verso Cosmos DB e il Business Layer
    return r.Logic.GetFridgeById(ctx, id)
}

// Endpoint vitalità per l'orchestratore
http.HandleFunc("/health", func(w http.ResponseWriter, r *http.Request) {
    w.Write([]byte("healthy"))
})
```





Perché GraphQL?

A differenza delle API REST tradizionali, **GraphQL** permette al frontend di MOCC di richiedere solo la fetta esatta di dati necessaria per ogni vista. Questo riduce il traffico di rete e il numero di chiamate, migliorando drasticamente la reattività della Mobile App.

Strong Typing: Ogni entità (Fridge, Recipe, User) è rigorosamente definita nello schema, eliminando ambiguità tra backend (Go) e client (Flutter).

Single Endpoint: Tutta la logica è accessibile tramite un unico punto di ingresso via APIM, semplificando la gestione di certificati e firewall.

```
type Query {  
    me: User!  
    myFridges: [Fridge!]!  
    recipe(id: ID!): Recipe!  
}  
  
type Mutation {  
    consumeInventoryItem(id: ID!, amount: Float!): InventoryItem!  
    createRecipe(input: CreateRecipeInput!): Recipe!  
    cookRecipe(id: ID!): Recipe!  
    addComment(postId: ID!, text: String!): Comment!  
    generateUploadSasToken(filename: String!, purpose: UploadPurpose!): String!  
}
```





Azure Functions App - Librerie

Le funzioni implementano la logica reattiva del sistema utilizzando **Python**. Gestiscono le operazioni pesanti disaccoppiate dalle chiamate API dirette, processando dati in background non appena disponibili nello storage.

Analisi Scontrini: Sfrutta la libreria azure-ai-documentintelligence per convertire immagini di ricevute in liste di prodotti strutturate.

Generazione Ricette: Orchestrazione di Azure OpenAI per trasformare gli ingredienti disponibili in suggerimenti culinari personalizzati.

Moderazione Social: Controllo automatizzato di testi e immagini caricate dagli utenti per garantire conformità e sicurezza.

Gestione Scadenze: Task programmati che analizzano l'inventario e attivano l'invio di notifiche push prima che i prodotti scadano.



GitHub Action

Secret Injection

Configurazione automatica di segreti e credenziali

Functions Deployment

Rilascio istantaneo della logica serverless in Python.

Revision Rolling

Aggiornamento atomico dei servizi senza interruzioni.

Backend Deployment

Build Docker Go e aggiornamento revisioni su ACA.

Static Web Deployment

Compilazione e hosting del bundle Flutter Web.



04.

CONCLUSIONI

Analisi delle limitazioni e proposte di sviluppo

Limitazioni di MOCC

L'architettura **cloud-native** garantisce scalabilità e **costi ridotti** mediante standard di sicurezza integrati. Sono tuttavia presenti margini di miglioramento verso configurazioni di rete e **governance più robuste**.

Vantaggi

- Modello di costo **Pay-per-use**
- Deploy infrastrutturale automatizzato
- Sicurezza **Keyless** (Managed Identity)
- Integrazione nativa AI & OCR
- Unified Access Layer (APIM & GraphQL)

Svantaggi

- Latenze da **Cold Start** (Piani Consumption)
- Assenza di Application Gateway e **WAF**
- Mancanza di isolamento in Virtual Network (VNET)
- Throughput **Cosmos DB limitato** (Cap 400 RU/s)
- Redis Enterprise senza High Availability (SKU B0)
- Assenza di **test E2E** nella pipeline CI/CD



Grazie!



Dubbi o curiosità?
m.cosenza11@studenti.unisa.it



[mariocosenza/mocc](https://github.com/mariocosenza/mocc)



Mario Cosenza

