



DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Applicazioni Cloud-Native

vantaggi e svantaggi delle architetture serverless

Relatore: **Prof. Mauro Migliardi**

Laureando: **Federico Masiero**

ANNO ACCADEMICO 2021/2022

Data di laurea: 18 Ottobre 2022

Abstract

Progettare, sviluppare e mantenere applicativi è un compito arduo. Le competenze che ogni individuo di un team deve acquisire sono sempre maggiori, le figure coinvolte in un progetto sono sempre di più e le applicazioni devono soddisfare determinati requisiti in ambito di sicurezza, performance, distribuzione e scalabilità. Lo scenario appena descritto ha come principale risultato quello di alzare la barriera all'ingresso per tutte quelle aziende che vogliono migliorare la loro offerta commerciale attraverso l'utilizzo di tecnologie e software proprietari.

Un aspetto molto importante da tenere in considerazione quando si progettano applicativi software, è la scalabilità delle risorse. Infatti, per far fronte ad esigenze circoscritte, ad esempio picchi di utilizzo in determinate fasce orarie del giorno, si è costretti a sovradimensionare alcune risorse. Inoltre, si rende necessario ridurre il “*time-to-market*”, ottimizzare l'effort degli sviluppatori e semplificare la manutenzione futura dell'applicazione.

Per far fronte alle esigenze appena descritte, sono state introdotte nel mercato tecnologie di *Serverless Computing* con l'obiettivo di fornire un servizio altamente accessibile, scalabile, performante e relativamente automatico. Le tecnologie di *Serverless Computing* sono soluzioni *Platform as a Service* (PaaS) in cui il fornitore di servizi cloud si occupa di allocare e gestire le risorse necessarie per eseguire l'applicazione. Questo consente al team impegnato nello sviluppo di concentrarsi sul software piuttosto che sulla manutenzione dei server. Le tecnologie appena descritte riducono lo spreco di risorse aumentando o diminuendo l'allocazione a seconda del traffico. In questo modo si pagherà solamente per le risorse effettivamente utilizzate dall'applicazione.

La presente tesi ha come principale obiettivo quello di approfondire le tecnologie serverless analizzando la letteratura e gli strumenti disponibili, con lo scopo di identificare vantaggi e limiti nell'utilizzo. Si vedrà, inoltre, l'applicazione pratica di queste tecnologie nella realizzazione di un applicativo cloud-native per la gestione di un concorso di opere d'arte. Il fornitore di servizi cloud scelto è Amazon con la sua piattaforma AWS.

Elenco delle figure

2.1	Infrastruttura con virtualizzazione	6
2.2	Infrastruttura con containerizzazione	7
2.3	Elementi chiave del modello di elaborazione serverless	10
3.1	Esempio del funzionameno di <i>Amazon SNS</i> [22]	20
4.1	Architettura Software Contest111	53
4.2	Flusso Generale	55
4.3	Flusso Login e Registrazione	56
4.4	Flusso caricamento opere d'arte e iscrizione concorsi	57
4.5	Flusso per la votazione	58
5.1	Roadmap del progetto Contest111	63
5.2	Sviluppo: DynamoDB - Global Secondary Index	68
A.1	Mockup Design - Lista opere d'arte	75
A.2	Mockup Design - Dettagli opera d'arte	76
A.3	Mockup Design - Crea opera d'arte	77
A.4	Mockup Design - Dettagli concorso	78
A.5	Mockup Design - Home Page	79

Elenco delle tabelle

4.1	AdR: Glossario dei termini	27
4.2	AdR: Login e Registrazione	29
4.3	AdR: Opere d'arte	31
4.4	AdR: Artisti	32
4.5	AdR: Concorsi	33
4.6	AdR: Profilo e Impostazioni	34
4.7	AdR: Home Page	35
4.8	AdR: Menù	36
4.9	AdR: Votazione	37
4.10	AdR: Dashboard di amministrazione	38
4.11	Modello dei dati: USER	41
4.12	Modello dei dati: ARTWORK	42
4.13	Modello dei dati: CONTEST	43
4.14	Modello dei dati: SUBSCRIPTION	44
4.15	Modello dei dati: VOTE	45
4.16	DynamoDB - Global Secondary Index	52
4.17	Parametri per la stima di costo del cloud	59

Elenco dei codici

4.1	DynamoDB Element - USER	47
4.2	DynamoDB Element - ARTWORK	48
4.3	DynamoDB Element - CONTEST	49
4.4	DynamoDB Element - SUBSCRIPTION	50
4.5	DynamoDB Element - VOTE	51
5.1	Definizione Amazon Cognito in AWS SAM	66
5.2	Policy di accesso Amazon S3	67
B.1	GraphQL Schema - Artwork	80
B.2	GraphQL Schema - Subscription	82
B.3	GraphQL Schema - Vote	82
B.4	GraphQL Schema - Mutation	82
B.5	GraphQL Schema - Query	83
C.1	Funzioni VTL - createArtwork	85
C.2	Funzioni VTL - subscribeToContest	86
C.3	Funzioni VTL - voteArtwork	86
C.4	Funzioni VTL - checkExistingSubscription	87

Indice

Abstract	i
1 Introduzione	1
1.1 Scopo della tesi	2
1.2 Metodologia	2
1.3 Schema della tesi	2
2 Serverless Computing	4
2.1 Cos'è <i>Serverless</i>	5
2.2 Evoluzione del <i>Cloud Computing</i>	6
2.3 PaaS (<i>Platform as a Service</i>)	8
2.4 FaaS (<i>Function as a Service</i>)	9
3 Panoramica dei servizi AWS	12
3.1 Servizio di autenticazione: <i>Amazon Cognito</i>	13
3.2 Database NoSQL: <i>Amazon DynamoDB</i>	14
3.3 Archiviazione di oggetti: <i>Amazon S3</i>	15
3.4 Computazione on-demand: <i>AWS Lambda</i>	16
3.5 Endpoint API REST e WebSocket: <i>AWS API Gateway</i>	17
3.6 Backend GraphQL: <i>AWS AppSync</i>	18
3.7 Hosting server: <i>AWS Amplify</i>	19
3.8 Messaggistica pub/sub: <i>Amazon SNS</i>	20
3.9 Code di messaggi: <i>Amazon SQS</i>	21
3.10 Monitoraggio: <i>Amazon CloudWatch</i>	22
3.11 Infrastructure-as-a-Code: <i>Amazon CloudFormation</i>	23
3.12 Framework Serverless: <i>AWS SAM</i>	24
4 Progetto	25
4.1 Analisi dei requisiti	26
4.1.1 Glossario dei termini	27
4.1.2 Login e Registrazione	28
4.1.3 Opere d'arte	30
4.1.4 Artisti	32

4.1.5	Concorsi	33
4.1.6	Profilo e Impostazioni	34
4.1.7	Home Page	35
4.1.8	Menù	36
4.1.9	Votazione	37
4.1.10	Dashboard di amministrazione	38
4.2	Progettazione	39
4.2.1	Modello dei dati	40
4.2.2	DynamoDB - Single Table Design	46
4.2.3	Architettura dell'applicativo	53
4.2.4	Flussi di utilizzo dell'applicazione	54
4.2.5	Previsione di costo del cloud	59
5	Sviluppo	61
5.1	Design mockup	62
5.2	Project Management	63
5.3	Back End	65
5.3.1	Amazon Cognito	66
5.3.2	Amazon S3	67
5.3.3	Amazon DynamoDB	68
5.3.4	AWS AppSync	69
5.4	Front End	71
5.5	Sviluppi futuri	72
6	Conclusioni	73
A	Mockup Design	75
B	GraphQL Schema	80
C	Funzioni VTL	85
	Bibliografia	89

Capitolo 1

Introduzione

Numerose aziende stanno traendo vantaggio dall'esecuzione delle loro applicazioni nel cloud. La migrazione in cloud delle architetture applicative esistenti mira a ridurre il costo totale dell'infrastruttura (*TCO - Total Cost of Ownership*) e migliorare il *time-to-market*. Rispetto a soluzioni locali, il cloud semplifica notevolmente la creazione, l'implementazione e la gestione di server e delle applicazioni che vengono eseguite su di essi. Tutto questo si traduce in un risparmio sui costi, derivante dal modello di fatturazione *pay-as-you-go*, e in una maggiore agilità nell'utilizzo di risorse IT on-demand.

Qualsiasi architettura basata su server, anche se in cloud, richiede di essere progettata per garantire scalabilità e affidabilità. Inoltre, le aziende devono affrontare sempre nuove sfide legate all'applicazione di *patch* man mano che i loro applicativi si evolvono. Infine, devono tenere conto dei picchi di carico e quindi tentare di ridimensionare la propria infrastruttura quando e dove possibile per ridurre i costi, il tutto proteggendo l'esperienza degli utenti finali e l'integrità dei sistemi. In questo contesto, i server inattivi e sottoutilizzati si rivelano costosi e dispendiosi. Una ricerca commissionata da *AWS* a *451 Research* ha stimato che l'utilizzo medio dei server è del 18% circa. [18]

Serverless è una soluzione di distribuzione in cui, invece di avere server in esecuzione continuamente, le istanze dell'applicazione vengono eseguite solo a seguito di eventi specifici. Utilizzando servizi serverless, sviluppatori e progettisti possono concentrarsi sulla logica del prodotto e della gestione delle richieste/eventi, mentre il provider dell'infrastruttura si occupa della pianificazione della capacità, della pianificazione delle attività e del monitoraggio operativo. [26]

Di conseguenza, le aziende che adottano questo tipo di soluzione possono ottenere un *time-to-market* più rapido con cicli di sviluppo, implementazione e test più brevi. Inoltre, la riduzione delle spese generali di gestione del server riduce il TCO, che in definitiva si traduce in vantaggi competitivi.

Poiché le piattaforme serverless sono ancora agli albori, esistono limitazioni significative per molti casi d'uso. Inoltre, le aziende non sono pienamente consapevoli dei modelli, dei vantaggi e delle sfide che derivano da applicazioni con architettura Serverless. Alcune delle principali problematiche che possono essere riscontrate sono: SLA (*Service-Level Agreement*), tempo di inattività, latenza nell'esecuzione, non conformità ad alcuni standard, durata di esecuzione relativamente breve, nessun ambiente locale ed una forte dipendenza dal fornitore. [19]

1.1 Scopo della tesi

L'elaborazione serverless ha molto da offrire ai progettisti, agli sviluppatori e ai proprietari di applicativi software. Chiaramente, non è un approccio universale e una corretta implementazione dipende da molti fattori e importanti considerazioni. Lo scopo di questa tesi è approfondire il modello di programmazione delle applicazioni serverless, descrivere i vantaggi e le sfide di questa tecnologia, analizzare la produttività dello sviluppo, le prestazioni e la scalabilità. Attraverso il materiale presentato si potrà capire quando l'architettura serverless è la scelta giusta, così da evitare insidie comuni quando si lavora con questo stack tecnologico.

1.2 Metodologia

Per la stesura di questa tesi si è reso necessario, dapprima, uno studio approfondito della letteratura disponibile e, successivamente, un approccio pratico allo sviluppo di un'applicazione. Ci si è velocemente resi conto, però, che le informazioni disponibili sono limitate e, talvolta, obsolete a causa della rapida evoluzione della tecnologia. Per lo studio della letteratura si è utilizzato materiale da ricerche precedenti, whitepaper, webinar, conferenze e blog di settore. Per lo sviluppo dell'applicazione si sono utilizzate le guide e la documentazione di AWS, oltre ad innumerevoli blog di settore.

1.3 Schema della tesi

Questo elaborato si compone di quattro parti principali. La prima parte, presentata nel capitolo 2, è un approfondimento teorico sulle tecnologie serverless. La seconda parte, capitolo 3, presenta alcuni dei servizi serverless della piattaforma AWS. La terza parte, presentata nel capitolo 4, presenta la progettazione di un'applicazione che utilizza tecnologie serverless. L'ultima parte, capitolo 5, descrive l'approccio pratico utilizzato per la realizzazione dell'applicativo software. Infine, si riassumono

i risultati del progetto, si traggono alcune conclusioni e si forniscono suggerimenti per un possibile lavoro futuro.

Capitolo 2

Serverless Computing

Il “*Serverless Computing*”, o più semplicemente *Serverless*, è un trend in rapida crescita nel mondo delle architetture software. Le “*Big Three*” del *Cloud Computing* (Amazon, Google e Microsoft), infatti, stanno investendo molto in questa nuova tecnologia. Come spesso accade per i nuovi trend non è presente una definizione ufficiale. Infatti, in accordo con quanto indicato nella pubblicazione “*Serverless Computing: Current Trends and Open Problems*” [3], il termine “*Serverless*” è stato coniato dall’industria tecnologica la quale ne determina la sua continua evoluzione.

In questo capitolo verrà dapprima affrontata una panoramica generale dell’evoluzione del cloud computing. Poi, verrà approfondita la tecnologia serverless nelle sue tipologie: PaaS e FaaS con i relativi vantaggi e limiti di implementazione.

2.1 Cos'è *Serverless*

Due delle più appropriate definizioni, che sono state individuate nella letteratura, sono le seguenti:

- “*Serverless computing* è un termine coniato dall’industria per descrivere un modello di programmazione e un’architettura in cui piccoli frammenti di codice vengono eseguiti nel cloud senza alcun controllo sulle risorse su cui viene eseguito il codice.” [3]
- “*Il serverless computing* è una forma di *cloud computing* che consente agli utenti di eseguire applicazioni basate su eventi e fatturate in modo granulare.” [9]

Il termine “*Serverless*” potrebbe risultare fuorviante ed è importante precisare che non si riferisce all’assenza di server nel cloud computing, ma all’assenza di gestione dell’infrastruttura da parte del team. Il *serverless computing*, infatti, astrae la maggior parte dei problemi relativi alle risorse (e.g. provisioning eccessivo/insufficiente, tolleranza agli errori, scalabilità, distribuzione) in modo tale che gli sviluppatori non debbano pensare al server su cui verrà distribuito il loro codice. [3]

Mike Roberts, consulente e autore del libro “*Programming AWS Lambda*” [12], identifica due diversi tipi di implementazioni del “*serverless computing*”: [20]

- **PaaS (*Platform as a Service*)**: è l’implementazione di componenti server-side, generalmente utilizzati da applicativi software, tramite servizi esterni pronti all’uso. Questi servizi di terze parti, solitamente, espongono le loro funzionalità tramite API. Alcuni dei più comuni servizi sono i cluster di database gestiti (e.g. *Google Firebase*) o componenti di autenticazione (e.g. *Auth0* e *Amazon Cognito*).
- **FaaS (*Functions as a Service*)**: è un metodo per la creazione e la distribuzione di applicazioni server-side basato su singole funzioni come unità di distribuzione. I servizi di FaaS più conosciuti sono *AWS Lambda*, *Google Cloud Functions*, *IBM Cloud Functions*, *Microsoft Azure Functions* e *Apache OpenWisk*.

2.2 Evoluzione del *Cloud Computing*

Negli ultimi decenni, lo sviluppo del software e la gestione delle infrastrutture hanno subito diverse evoluzioni. Nella breve panoramica che segue, vengono descritte le innovazioni tecnologiche che hanno determinato questo cambiamento.

L'utilizzo di server dedicati, ossia macchine fisiche installate e gestite localmente, era lo standard di settore per le implementazioni di infrastrutture tecnologiche. Sebbene questo metodo offra un elevato livello di sicurezza e disponibilità, era molto comune che le infrastrutture fossero ampiamente sovradimensionate, con un conseguente spreco di risorse e denaro.

Verso la fine degli anni '90, la tecnologia di virtualizzazione ha rivoluzionato l'implementazione delle infrastrutture software, disaccoppiando le applicazioni dall'hardware sottostante. I server virtuali (*Virtual Machine*) vengono eseguiti su server dedicati, le cui risorse sono condivise. Con l'introduzione di questa tecnologia, si è riusciti a ridurre notevolmente lo spreco di risorse ed i costi generali dell'infrastruttura. Proprio per questo motivo ha preso piede velocemente e molte aziende iniziarono ad adottarla nelle loro infrastrutture.

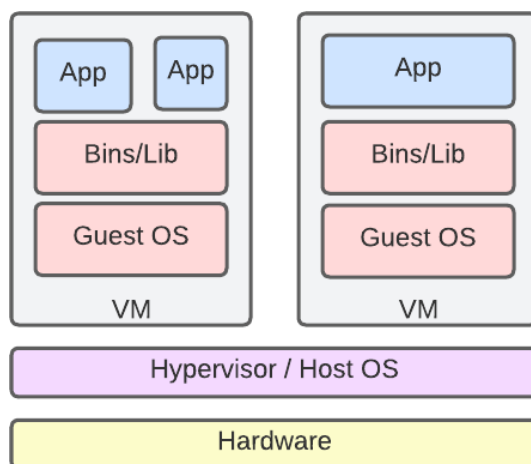


Figura 2.1: Infrastruttura con virtualizzazione

All'inizio del 2006, *Amazon Web Services* (AWS) si è rilanciato nel mercato come piattaforma che offre spazio di elaborazione e archiviazione on-demand a sviluppatori e aziende. Con l'avvento della virtualizzazione anche i profili professionali subirono un'evoluzione significativa, passando da amministratori di sistema a DevOps (*Development & Operations*) che si occupano del provisioning delle macchine virtuali. Sebbene la IaaS (*Infrastructure as a Service*) abbia risolto molti problemi relativi al provisioning dell'infrastruttura, i sistemi e il carico delle applicazioni sono rimasti

comunque indipendenti con una conseguente inefficienza nell'utilizzo delle risorse. [8]

I server virtuali sono stati per molti anni la spina dorsale delle infrastrutture tecnologiche aziendali. Un punto di svolta nel mondo della virtualizzazione è stata la diffusione della containerizzazione. I container sono ambienti di elaborazione che combinano varie risorse e le isolano dal resto del sistema, con lo scopo di incapsulare un'applicazione e le sue dipendenze all'interno di essi. In questo modo le risorse non vengono utilizzate per l'esecuzione di attività dedicate ai sistemi operativi. Infatti, il container dell'applicazione è indipendente dal sistema operativo su cui viene eseguito.

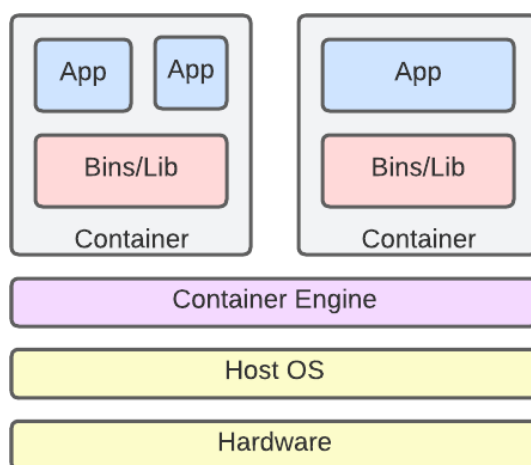


Figura 2.2: Infrastruttura con containerizzazione

Uno dei principali vantaggi dei container è la riproducibilità ed il trasferimento da un sistema all'altro, senza influire sulle funzionalità dell'applicazione o dover reinstallare tutte le dipendenze sulla nuova macchina. L'ambiente può essere facilmente riprodotto anche in ambito locale. Tutto ciò ha reso il ciclo di sviluppo/test più efficiente. [16]

Infine, negli ultimi anni, un nuovo paradigma ha fatto il suo ingresso nel panorama delle infrastrutture tecnologiche: il *serverless computing*. Come accennato precedentemente, possiamo riassumere il concetto alla base di questa tecnologia in questo modo: *“Una forma di cloud computing che elimina la gestione dei server, consentendo il ridimensionamento e la fatturazione granulari, supportando un'architettura software basata su eventi che utilizza piccole funzioni come singole unità di distribuzione.”*

2.3 PaaS (*Platform as a Service*)

Platform as a Service (PaaS) è un modello di cloud computing in cui un provider di terze parti fornisce strumenti hardware e software alle aziende, solitamente utilizzati per lo sviluppo di applicazioni. Queste tipologie di piattaforme possono includere servizi di elaborazione, memoria, archiviazione, database e altri servizi utili allo sviluppo di applicazioni. Un provider di PaaS ospita l'hardware e il software sulla propria infrastruttura sollevando le aziende dall'onere di doverli gestire. Inoltre, per migliorare l'esperienza del team di sviluppo, spesso, il provider fornisce anche un'interfaccia utente. [2]

Gli scenari più comuni in cui vengono utilizzate delle PaaS sono: [14]

- **Framework di sviluppo.** Consente agli sviluppatori di creare applicazioni utilizzando componenti software integrati, dove sono incluse funzionalità cloud come scalabilità, disponibilità elevata e capacità multi-tenant.
- **Analisi dei dati o BI (*Business Intelligence*).** Consente alle organizzazioni di analizzare i propri dati migliorando le previsioni e i risultati delle decisioni aziendali.

I principali vantaggi delle PaaS sono la semplicità di utilizzo e la convenienza economica. Questi servizi vengono spesso utilizzati da aziende in fase di startup, consentendo di accedere a risorse all'avanguardia ad un prezzo ragionevole. Infatti, la possibilità di pagare su base ricorrente o per l'utilizzo effettivo, consente alle aziende di avere un modello di pricing chiaro e predittivo. Infine, l'utilizzo delle PaaS delega al provider esterno la responsabilità di fornire, gestire e aggiornare le risorse in questione. [32]

Sebbene i vantaggi delle PaaS siano molteplici ed evidenti, si deve considerare anche il rovescio della medaglia. Alcune delle sfide da affrontare, quando si sceglie di utilizzare un servizio PaaS, possono essere: la dipendenza dal fornitore del servizio (*Lock-In*) e la compatibilità con piattaforme preesistenti. Infine, mentre i provider sono responsabili per la sicurezza dell'infrastruttura e delle risorse messe a disposizione, le aziende sono responsabili della sicurezza delle applicazioni che vengono sviluppate. [11]

2.4 FaaS (*Function as a Service*)

Function as a service (FaaS) è un modello di cloud computing che consente di eseguire una singola funzione o parte di un'applicazione, senza dover gestire l'infrastruttura server. FaaS è basato sugli eventi e viene eseguito in container stateless appositamente creati, permettendo così un modello di pricing *pay-as-you-go*. Il primo servizio FaaS è stato rilasciato da *hook.io* nel 2014, seguito da *AWS Lambda*, *Google Cloud Functions*, *Microsoft Azure Functions*, *IBM/Apache's OpenWhisk* e *Oracle Cloud Fn.* [31] [17]

Una funzione serverless è l'unità di distribuzione di base dei servizi FaaS. Queste funzioni sono tutte distribuite separatamente in container che possono essere istanziati individualmente nel momento in cui il provider riceve un evento che desidera utilizzare quella specifica funzione. Perseguendo l'approccio “*1 funzione - 1 container*” potrebbe esserci un enorme spreco di memoria. Per far fronte a questa possibilità i provider di servizi FaaS, AWS in questo caso specifico, utilizzano delle macchine virtuali con configurazione minima (*micro-VM*) ed un numero limitato di container istanziati in ognuna di esse [27]. Si può evincere, da un whitepaper pubblicato da AWS [25], che i container istanziati in queste *micro-VM* non sono i classici *Docker* o *Kubernetes*, ma hanno una configurazione minimale che utilizza tecnologie integrate nel *Kernel Linux*, assieme ad altre tecnologie di isolamento proprietarie.

Un'importante caratteristica delle funzioni serverless è che sono intrinsecamente *stateless*. Ciò è imposto dal ciclo di vita delle istanze che vengono fornite su richiesta e, ove possibile, rimosse. Perciò, qualsiasi stato temporaneo archiviato in memoria verrebbe eliminato al momento del *deprovisioning*. La proprietà stateless delle funzioni serverless garantisce che, fornito lo stesso input di dati, l'output prodotto sarà sempre lo stesso in qualsiasi momento. Ciò significa che la funzione in esecuzione non avrà informazioni su quelle precedentemente eseguite. Così facendo, il provider può pianificare l'esecuzione della funzione in qualsiasi nodo del cluster. Purtroppo, ci sono dei vincoli sulla tipologia di applicazioni che è possibile strutturare con questa logica. Infatti, non è sempre possibile costruire software senza uno stato persistente.

Le funzioni serverless possono essere scalate orizzontalmente in modo semplice e l'intero processo può essere eseguito in un tempo relativamente breve, sebbene dipenda da molti fattori quali il linguaggio di programmazione, le dipendenze e la memoria richiesta. La parte più dispendiosa, in termini di tempo, dell'intero processo è la creazione del container che andrà ad eseguire la funzione. Viene definito “*cold-start*” il tempo impiegato dal provider di servizi cloud per creare il container e avviare la funzione. La maggior parte dei provider, per alleviare questo ritardo, mantiene la funzione “*warm*” per un determinato periodo dopo l'utilizzo; in pratica il container

non viene eliminato subito dopo l'utilizzo, ma tenuto inattivo per un certo periodo. [30]

Gli elementi chiave del modello di elaborazione serverless sono: il Faas Controller, le origini degli eventi, le istanze delle funzioni e gli altri servizi forniti dal provider.

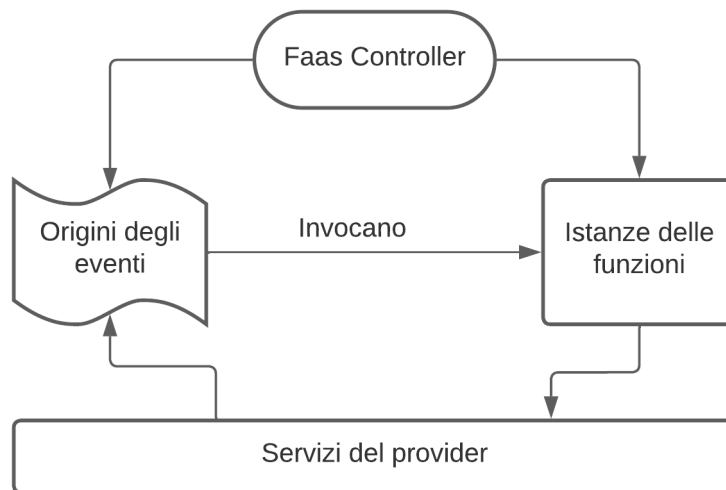


Figura 2.3: Elementi chiave del modello di elaborazione serverless

Il *FaaS Controller* è responsabile della gestione e del monitoraggio delle istanze delle funzioni. La gestione include l'esecuzione del codice, il ridimensionamento delle istanze in base alle richieste e il loro deprovisioning quando sono inattive. Le origini degli eventi possono attivare una o più istanze della funzione. Le origini più comuni sono le richieste HTTP, con la chiamata all'endpoint della funzione implementata, e l'esecuzione su base temporale in cui lo sviluppatore può scegliere di attivare la funzione in modo pianificato.

La funzione eseguita riceve, come input, i dati dell'evento e le informazioni sul contesto. I dati dell'evento dipendono dall'origine, mentre il contesto fornisce informazioni sulle risorse (e.g. limiti di memoria o tempo) e sull'ambiente di esecuzione (e.g. variabili globali e di ambiente). La funzione, infine, può comunicare con altri servizi del provider che, a loro volta, possono emettere nuovi eventi.

Una delle caratteristiche più interessanti del servizio FaaS è la politica di pricing "*pay-as-you-go*". Generalmente, i provider dei servizi di cloud computing addebitano l'organizzazione in base alla quantità di memoria riservata ed al tempo di esecuzione in relazione al numero di chiamate ricevute dalla funzione. Se viene confrontato questo modello di pricing con quello di una soluzione *Infrastructure as a Service* (IaaS), si può dedurre facilmente che FaaS non è la scelta giusta per tutte le

applicazioni. Si può, quindi, ottenere un risparmio nel caso di applicazioni a carico elevato, ma variabile. [4]

Sebbene l'elaborazione serverless possa sembrare il “*Sacro Graal*” delle soluzioni di distribuzione, è un campo in rapida crescita e tutto da scoprire. Questa tecnologia presenta diverse carenze che la rendono inadatta per specifiche tipologie di applicazioni. Alcune delle sfide più importanti sono di seguito analizzate. [19]

Service-Level Agreement (SLA). Gli attuali servizi FaaS messi a disposizione sono relativamente nuovi ed i provider non offrono ancora uno SLA operativo o di uptime. Occasionalmente, dunque, gli utenti potrebbero riscontrare tassi di errore o latenze superiori al previsto. Ciò rende gli attuali servizi FaaS non adatti ad attività *mission-critical*.

Latenza potenzialmente elevata. Il servizio FaaS decide autonomamente quando aumentare o diminuire il numero di istanze attive e gli sviluppatori non hanno alcun controllo su questo processo. Solitamente un'istanza inattiva può essere nuovamente utilizzata entro alcuni minuti, ma verrà eliminata dopo un periodo di inattività più lungo. Ciò significa che i servizi utilizzati molto di rado potrebbero subire una latenza elevata e costante. Anche per i servizi utilizzati di frequente gli utenti potrebbero riscontrare una certa latenza aggiuntiva, soprattutto durante i picchi di utilizzo quando vengono create molte nuove istanze.

No compliance. Alcune applicazioni necessitano di essere conformi a vari standard di sicurezza governativi e di settore. La maggior parte dei servizi FaaS attualmente presenti nel mercato non presenta alcuna conformità agli standard più diffusi (e.g. SOC, PCI, HIPAA BAA and FedRAMP).

Durata relativamente breve. Le funzioni serverless hanno un tempo massimo di esecuzione imposto dal provider e non c'è modo per gli sviluppatori di estendere tale limite. Dunque ogni singola attività deve essere completata entro quel lasso di tempo. Alcuni provider offrono soluzioni per concatenare e collegare più funzioni, ma ciò non rimuove il limite di tempo per una singola attività.

Ambiente di esecuzione locale. Sebbene possa essere istanziato localmente un container ed eseguita la funzione, non esiste un modo semplice di eseguire un ambiente simulato su una macchina locale per scopi di sviluppo o test.

Lock-In del fornitore. Lavorare in un ambiente serverless può rendere il codice dell'applicazione fortemente dipendente dal provider di servizi di cloud computing.

Capitolo 3

Panoramica dei servizi AWS

Il provider di servizi di cloud computing scelto per questa tesi e per il progetto realizzato è *Amazon Web Services* (AWS). Esistono molti servizi forniti da AWS che possono essere combinati per creare un'applicazione serverless. Una lista non esaustiva viene riportata di seguito.

- Servizio di autenticazione: *Amazon Cognito*
- Database NoSQL: *Amazon DynamoDB*
- Archiviazione di oggetti: *Amazon S3*
- Computazione on-demand: *AWS Lambda*
- Endpoint API REST e WebSocket: *AWS API Gateway*
- Backend GraphQL: *AWS AppSync*
- Hosting server: *AWS Amplify*
- Messaggistica pub/sub: *Amazon SNS*
- Code di messaggi: *Amazon SQS*
- Monitoraggio: *Amazon CloudWatch*
- Infrastructure-as-a-Code: *Amazon CloudFormation*
- Framework Serverless: *AWS SAM*

Di seguito verranno approfonditi, uno per volta, i servizi sopra elencati.

3.1 Servizio di autenticazione: *Amazon Cognito*

Amazon Cognito è un servizio PaaS. Fornisce servizi di autenticazione, autorizzazione e gestione degli utenti per le applicazioni web e mobile. Un utente potrà autenticarsi con il classico username / email e password o attraverso un servizio di terze parti quali Facebook, Google, Apple, ecc... I due componenti principali di *Amazon Cognito* sono i pool di utenti e i pool di identità. I pool di utenti sono directory utente che forniscono opzioni di registrazione e accesso. I pool di identità ti consentono di concedere ai tuoi utenti l'accesso ad altri servizi AWS. È possibile utilizzare pool di identità e pool di utenti separatamente o insieme. *Amazon Cognito* è disponibile in più *AWS Region* in tutto il mondo. In ogni regione, *Amazon Cognito* è distribuito su più *Availability Zones*. Le *Availability Zones* sono fisicamente isolate l'una dall'altra, ma sono unite da connessioni di rete private, a bassa latenza, ad alta velocità effettiva e altamente ridondanti. [22]

3.2 Database NoSQL: *Amazon DynamoDB*

Amazon DynamoDB è un database non relazionale (*NoSQL*) del tipo Chiave/Valore (*Key/Value*) progettato nel 2007 da Amazon per un utilizzo interno. Attualmente è un servizio PaaS e, come tale, consente di alleggerire gli oneri di gestione legati al funzionamento e alla scalabilità. Poiché si tratta di un servizio di database serverless, gli utenti possono aumentare la capacità e il throughput modificando RCU (*Read Capacity Unit*) e WCU (*Write Capacity Unit*) in modo semplice ed istantaneo. Le sue caratteristiche principali sono: dati eventualmente consistenti e tolleranza alle partizioni, infatti soddisfa la combinazione AP del teorema CAP. [13] L'unità base di *DynamoDB* sono le tabelle. Ogni tabella contiene almeno una colonna obbligatoria, chiamata *partition key*, che viene utilizzata per identificare univocamente l'elemento inserito nella tabella. Inoltre, ogni elemento inserito può contenere ulteriori dati. [22]

3.3 Archiviazione di oggetti: *Amazon S3*

L'archiviazione di oggetti nel cloud, comunemente chiamata “*Public Cloud Storage*”, è un servizio che permette di memorizzare dati online tramite un provider che li gestisce. La modalità di erogazione del servizio è on-demand, perciò è possibile ottenere capacità e costi commisurati alle esigenze, così da garantire agilità, scalabilità a livello globale e durabilità, con accesso ai dati garantito indipendentemente dal momento e dalla provenienza della richiesta.

I requisiti per una corretta ed efficace memorizzazione dei dati nel cloud sono molteplici, ma tra i più importanti troviamo:

- **Durabilità.** I dati devono essere memorizzati in modo ridondante, idealmente in più strutture e su più dispositivi in ogni struttura. Non devono esserci perdite di dati a causa di calamità naturali, errore umano o guasti meccanici.
- **Disponibilità.** Tutti i dati devono essere disponibili quando occorrono. L'archiviazione nel cloud ideale offre un buon equilibrio fra i tempi di recupero e il costo.
- **Sicurezza.** Idealmente tutti i dati devono essere crittografati, sia quelli inattivi sia quelli in transito. I permessi e i controlli degli accessi devono funzionare nel cloud nello stesso modo in cui funzionano nello storage locale.
- **Backup e ripristino.** L'archiviazione nel cloud offre costi contenuti, durabilità elevata e scalabilità estrema per le soluzioni di backup e ripristino.

Il servizio PaaS “*Simple Storage Service*”, comunemente chiamato “*Amazon S3*”, è un *public cloud storage* che permette di salvare oggetti composti dai rispettivi dati e metadati, offrendo inoltre molte funzionalità aggiuntive che lo rendono uno dei leader di mercato. Un esempio di caso d'uso di *Amazon S3* è l'archiviazione del codice sorgente *AWS Lambda* di un'applicazione serverless. Inoltre, *Amazon S3* può anche essere configurato come origine di eventi per *AWS Lambda*. [22]

3.4 Computazione on-demand: *AWS Lambda*

AWS Lambda è un servizio FaaS che consente l'esecuzione di codice senza eseguire il provisioning o la gestione dei server. Può essere definito come il livello logico di un'applicazione serverless che fornisce funzionalità di elaborazione sotto forma di funzioni. Tali funzioni possono essere attivate da vari eventi che si verificano su AWS o su servizi di terze parti. Inoltre, segue il modello *pay-per-request*, come discusso nel capitolo precedente, il che significa che non ci sono costi aggiuntivi in assenza di eventi. Ogni volta che un evento viene emesso dall'origine a cui è collegata una funzione Lambda, questa verrà inizializzata. Il processo di inizializzazione include l'avvio dell'ambiente di esecuzione per la funzione, che di solito è un container. Dopo aver attivato il container, verranno installate le dipendenze software per il linguaggio di programmazione utilizzato dalla funzione. Quindi, viene scaricato il codice per la logica dell'applicazione, che solitamente è contenuto in *Amazon S3*. Infine, il codice verrà eseguito a seconda del tipo di evento e con i relativi parametri.

Un aspetto molto importante da tenere in considerazione quando si scrive una funzione su *AWS Lambda*, è la proprietà *stateless* dei servizi FaaS. Le origini degli eventi per una funzione Lambda possono seguire un modello *push* o un modello *pull*. Nel modello *push*, una funzione Lambda verrà eseguita ogni volta che si verifica un evento. D'altra parte, nel modello *pull*, *AWS Lambda* eseguirà periodicamente il *polling* dell'origine degli eventi e combinerà diversi eventi in un'unica chiamata di funzione.

[22]

3.5 Endpoint API REST e WebSocket: *AWS API Gateway*

Amazon API Gateway è un servizio PaaS che offre la possibilità di creare API REST e WebSocket. Il servizio funge da punto di ingresso al sistema ed è ampiamente utilizzato nelle applicazioni serverless come origine di eventi per le funzioni in *AWS Lambda*. [22]

3.6 Backend GraphQL: *AWS AppSync*

L'idea alla base di un'API GraphQL è che tutte le funzionalità dell'API possano essere disponibili in un singolo endpoint, tramite un linguaggio di query unificato (*Graph Query Language*). Il backend di un'applicazione è solitamente composto da più microservizi. Anziché effettuare richieste a vari endpoint per ottenere parti diverse dei dati necessari per creare una pagina Web, gli sviluppatori possono inviare una singola richiesta a un'API GraphQL. Questa è responsabile della raccolta di tutti i dati necessari dalle varie applicazioni e della restituzione del risultato. *AWS AppSync* è un servizio PaaS di API GraphQL che gestisce l'analisi e la risoluzione delle richieste, nonché la connessione ad altri servizi AWS come AWS Lambda, NoSQL e SQL database e API HTTP.

AWS AppSync è costituito da un proxy GraphQL, un servizio che riceve e analizza tutte le richieste GraphQL, e da più sottosistemi per la gestione di ogni specifico tipo di richiesta. I tipi di richiesta supportati sono *Query* (per ottenere dati dall'API), *Mutation* (per modificare i dati tramite l'API) e *Subscription* (connessioni di lunga durata per lo streaming di dati dall'API). L'interfaccia grafica consente agli sviluppatori di definire lo schema dell'API GraphQL e di connettere ciascuna funzione del resolver a un'origine dati. Le origini dati supportate per impostazione predefinita includono tabelle *Amazon DynamoDB*, database RDS, domini *Amazon Elasticsearch*, funzioni *AWS Lambda* ed endpoint HTTP di terze parti. Infine, è presente la possibilità di testare un'API GraphQL in tempo reale direttamente dall'interfaccia grafica di AWS. [22]

3.7 Hosting server: *AWS Amplify*

AWS Amplify Hosting è un servizio PaaS che fornisce un flusso di lavoro basato su git per l'hosting di applicazioni web con distribuzione continua. Supporta i framework SPA comuni, ad esempio *React*, *Angular*, *Vue.js*, *Ionic* ed *Ember*, nonché generatori di siti statici come *Gatsby*, *Eleventy*, *Hugo*, *VuePress* e *Jekyll*. Inoltre, una delle sue funzionalità più utili è quella di gestire gli ambienti di produzione e staging in base a diversi branch della repository. Grazie alle distribuzioni Atomic, *AWS Amplify* non presenta finestre di manutenzione durante il deploy di una nuova versione dell'applicativo, assicurandosi che venga aggiornata solo al termine dell'intera distribuzione. [22]

3.8 Messaggistica pub/sub: *Amazon SNS*

Simple Notification Service, comunemente conosciuto come *Amazon SNS*, è un servizio PaaS di messaggistica pub/sub fornito da AWS. Consente ai *publisher* (o *producer*) di inviare messaggi ai *subscriber* (o *consumer*). In questo servizio, i *publisher* non conoscono i *subscriber* e non interagiscono tra di loro direttamente, ma attraverso dei "topic". I *publisher* possono inviare messaggi nei *topic*, i quali verranno recapitati ai *subscriber* del medesimo *topic*. [22]

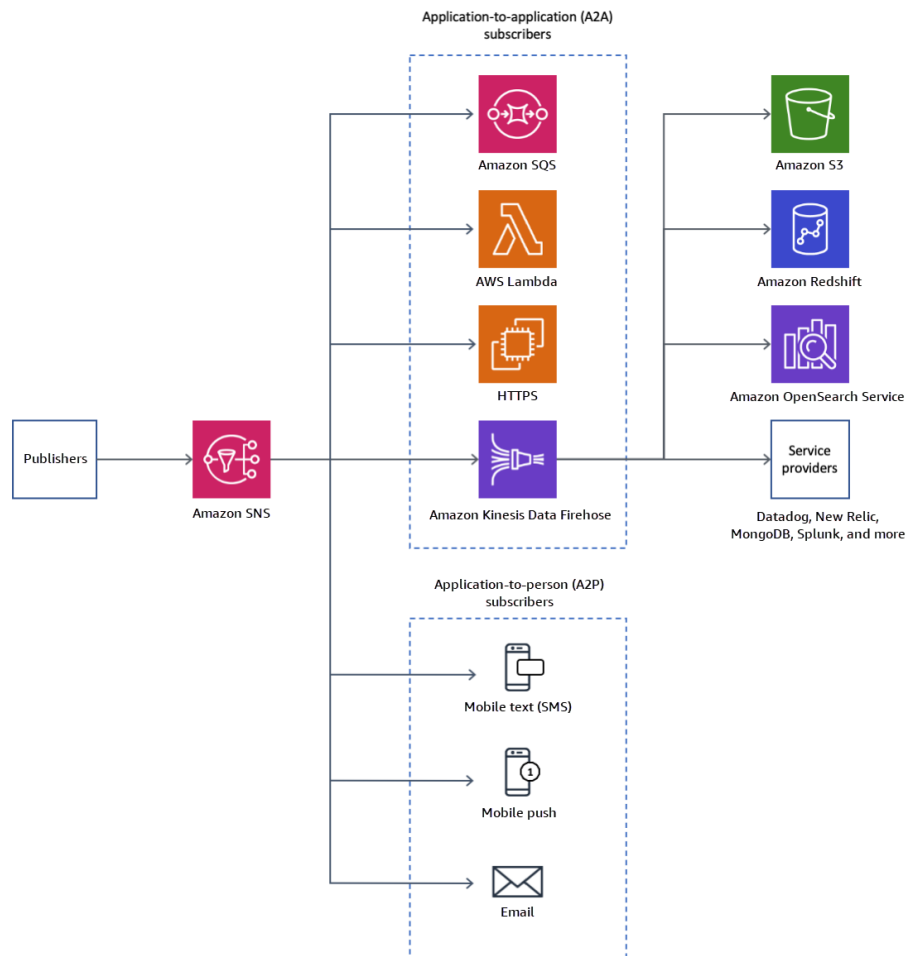


Figura 3.1: Esempio del funzionamento di *Amazon SNS* [22]

3.9 Code di messaggi: *Amazon SQS*

Simple Queue Service (Amazon SQS) è un servizio PaaS di accodamento della messaggistica. I messaggi vengono inviati dai mittenti alla coda, da cui i destinatari eseguono il polling. Dopo aver elaborato un messaggio, il destinatario deve eliminarlo dalla coda in modo che non venga elaborato due volte. In questo modo, un messaggio non può essere ricevuto da più destinatari. [22]

Amazon SQS offre due tipi di code di messaggi:

- **Code standard:** progettate per ottenere la massima velocità di trasmissione possibile e un delivery di tipo *at-least-once*.
- **Code FIFO:** progettate per garantire che i messaggi vengano elaborati esattamente secondo l'ordine in cui sono stati inviati.

3.10 Monitoraggio: *Amazon CloudWatch*

Amazon CloudWatch è un servizio PaaS di monitoraggio fornito da AWS. Utilizzando *CloudWatch*, si possono ottenere statistiche sullo stato del sistema, quasi in tempo reale. Log, parametri ed eventi possono essere raccolti dai vari servizi AWS e inviati ad *Amazon CloudWatch*. Ad esempio, i log emessi dalle funzioni *Lambda* possono essere inviati a *CloudWatch* a scopo di analisi. Inoltre, è presente la possibilità di impostare degli allarmi che si attivano quando viene violata una determinata condizione. Ad esempio, ogni volta che una funzione *Lambda* genera un errore. [22]

3.11 Infrastructure-as-a-Code: *Amazon CloudFormation*

Amazon CloudFormation è un servizio fornito da AWS che permette al team di sviluppo di configurare i componenti dell'infrastruttura in modo dichiarativo. *CloudFormation* permette alle aziende di adottare il modello *Infrastructure-as-a-Code* (IaaS). In questo tipo di modello i componenti dell'infrastruttura, come database e risorse di calcolo, sono dichiarati in un file. Il modello può quindi essere distribuito in AWS in cui diventa un *CloudFormation Stack* che andrà ad istanziare i componenti dell'infrastruttura. Un account AWS può avere molti *stack* e ogni *stack* può descrivere un insieme di più componenti. Per aggiornare uno *stack*, è necessario creare un *changeset* modificando il modello e caricandolo nuovamente in AWS. [22]

3.12 Framework Serverless: *AWS SAM*

AWS Serverless Application Model (AWS SAM) è un framework open-source che può essere utilizzato solo all'interno dell'ecosistema AWS. Nasce come estensione di *Amazon CloudFormation* per fornire un modo semplificato di definire le risorse serverless ed include due componenti principali: una specifica del modello e un'interfaccia a riga di comando (CLI). Il team di sviluppo può utilizzare un modello SAM per definire le risorse di uso comune per un'app serverless, come API (*Application Programming Interface*), funzioni e tabelle di database, come un unico stack, anziché distribuirle e gestirle manualmente e separatamente. I file che descrivono le risorse di un'applicazione serverless hanno formato JSON o YAML. La configurazione dei servizi attraverso il framework SAM include una sezione globale e una sezione per le risorse, origini degli eventi e proprietà. La sezione globale contiene la configurazione dell'ambiente (e.g. runtime, memoria, impostazioni VPC, variabili di ambiente, ecc. . .), mentre la sezione Risorse dichiara i servizi AWS che verranno inclusi nello stack (e.g. funzioni *Lambda*, *API Endpoint*, Tabelle *DynamoDB*). Grazie a *SAM Local*, un comando istanzia un container Docker e simula un gateway API e un ambiente Lambda nel sistema locale, uno sviluppatore può creare e testare le funzioni Lambda in un ambiente locale. [22]

Capitolo 4

Progetto

Per approfondire l'implementazione delle tecnologie serverless in un progetto reale, si prende a riferimento la realizzazione di un applicativo per la gestione di un concorso di opere d'arte. La piattaforma ha come obiettivo principale quello di coinvolgere gli artisti facendo caricare le loro opere d'arte e partecipare al contest dove in palio c'è un montepremi di 111.000 €. Con l'obiettivo di creare un network, la piattaforma coinvolgerà anche i fan (amici degli artisti ed amanti dell'arte in generale) dando loro la possibilità di votare le opere d'arte iscritte al contest. Considerata la fase di start-up del progetto e la sua imprevedibilità in termini di iscrizioni, caricamenti delle opere e visite da parte degli utenti, la soluzione proposta al cliente è lo sviluppo di un'applicazione web cloud-native con tecnologie serverless. In questo capitolo si andranno ad analizzare i requisiti dell'applicazione e quanto ideato durante la fase di progettazione.

4.1 Analisi dei requisiti

La start-up, grazie alla collaborazione con diverse gallerie d'arte e case d'asta, si rivolge fin dai primi giorni ad un pubblico di artisti internazionali. Ne consegue che l'applicazione dovrà essere disponibile in più lingue (inizialmente italiano ed inglese) e dovrà supportare più valute per i pagamenti.

Considerato il pubblico a cui si rivolge, si dovrà prestare particolare attenzione all'interfaccia grafica (UI) ed all'esperienza utente (UX). La UI dovrà essere semplice ed intuitiva ma con una particolare attenzione al design.

Inoltre, vista la quantità importante di immagini che saranno presenti in ogni pagina dell'applicazione si deve considerare attentamente i fattori di ottimizzazione e di caching per ottimizzare la velocità di caricamento. Proprio per la natura internazionale del progetto, l'intero stack tecnologico dovrà essere distribuito a livello globale in modo da ridurre al minimo la latenza.

Le transazioni di denaro all'interno della piattaforma dovranno essere gestite da un gateway di terze parti, il quale potrà assicurare velocità, sicurezza ed un'esperienza molto coinvolgente in fase di pagamento grazie al servizio di checkout gestito. Il servizio scelto è Stripe [28], che risponde appieno alle esigenze del progetto. Inoltre, si possono gestire agevolmente codici sconto, rimborsi e carrelli abbandonati.

Le metriche di successo, dal punto di vista tecnico, si possono riassumere in:

- Velocità delle piattaforma, misurabile dai seguenti parametri:
 - Time to First Byte (TTFB): si riferisce al tempo che intercorre tra la richiesta di una pagina da parte del browser e il momento in cui riceve il primo byte di informazioni dal server;
 - tempo di caricamento del contenuto above the fold;
 - tempo di caricamento totale;
 - peso della pagina.
- Ottimizzazione dei costi.

In questa sezione viene effettuata un'analisi approfondita di tutte le aree che andranno a comporre l'applicazione, analizzando i requisiti e gli aspetti tecnici d'implementazione.

4.1.1 Glossario dei termini

Termine	Descrizione
Admin — Amministratore	Utente con ruolo di amministratore. Può accedere all'area di amministrazione della piattaforma.
Artista	Utente con una o più opere d'arte caricate.
Contest — Concorso	È il concorso a cui gli artisti possono iscrivere una o più opere d'arte caricate in piattaforma.
Fan	Utente senza opere d'arte caricate.
Iscrizione	Partecipazione di un'opera d'arte al concorso, previo pagamento della quota d'iscrizione.
Opera d'arte	Entità chiave del progetto. Viene caricata da un utente che, da quel momento, assume il ruolo di Artista.
Utente	Utente registrato alla piattaforma. Può essere classificato come FAN o ARTISTA, in modo esclusivo.
Votazione	Processo di assegnazione di un voto, da parte di un utente, ad un'opera d'arte iscritta al concorso.
Voto	Valutazione assegnata durante il processo di votazione. È composto da un valore numerico e da un commento.

Tabella 4.1: AdR: Glossario dei termini

4.1.2 Login e Registrazione

Obiettivo: Fornire un'esperienza di registrazione e di accesso alla piattaforma sicura, intuitiva e funzionale. Il sistema di autenticazione verrà gestito dal servizio PaaS *AWS Cognito*.

Descrizione: Un utente può registrarsi alla piattaforma fornendo email, password e alcuni dati personali. Per confermare l'iscrizione alla piattaforma dovrà inserire un codice a 6 cifre che verrà inviato alla email registrata. L'utente potrà attivare l'autenticazione a due fattori (*MFA*) con l'utilizzo di un'applicazione (e.g. *Google Authenticator*). L'utente potrà effettuare il login alla piattaforma fornendo email e password. Se l'utente non ricorda la propria password, potrà recuperarla fornendo l'email ed il codice di verifica che verrà istantaneamente inviato. Se la *MFA* è attiva, durante la fase di login, sarà richiesta la conferma del codice. Dopo aver effettuato il primo accesso, verranno richiesti all'utente alcuni dati personali per finalizzare l'attivazione dell'account.

Assunzioni:

- Non sarà prevista la possibilità di usare la registrazione ed il login veloce attraverso *Federated Identity* (e.g. Google, Facebook, ecc...);
- Non c'è alcuna differenza di account tra Fan e Artisti.

Requisiti:

Requisito	Descrizione
Registrazione	Sarà possibile registrarsi fornendo email e password. La password dovrà rispettare determinate caratteristiche. Sarà necessario accettare, tramite una checkbox, le modalità per il trattamento dei dati personali. Infine, sarà possibile, attraverso una ulteriore checkbox, iscriversi alla newsletter.
MFA (<i>Multi Factor Authentication</i>)	Terminata la fase di registrazione, sarà possibile attivare l'autenticazione a due fattori (MFA). Se la MFA è abilitata, durante la fase di login, sarà necessario inserire il codice di conferma.
Login	Il login sarà effettuato con l'inserimento di email e password.
Recupero password	Se l'utente non ricorda la propria password, potrà recuperarla fornendo l'email di registrazione. Se l'email risulta registrata correttamente nella piattaforma, verrà inviato un codice di verifica. Sarà necessario fornire il codice di 6 cifre per procedere al reset della password.
Attivazione	Durante il primo accesso in piattaforma, verranno richiesti alcuni dati personali, quali nome, cognome e codice fiscale, per finalizzare l'attivazione dell'account.

Tabella 4.2: AdR: Login e Registrazione

4.1.3 Opere d'arte

Obiettivo: Dare la possibilità ad un artista di caricare e gestire le proprie opere d'arte. Dare la possibilità a tutti i visitatori di vedere le opere d'arte caricate nella piattaforma.

Descrizione: Ogni utente iscritto alla piattaforma, può caricare una o più opere d'arte. Se l'utente possiede almeno un'opera d'arte, sarà classificato come artista. Durante la fase di caricamento dell'opera si potrà decidere di iscriverla direttamente al concorso. Tutti i visitatori, anche se non registrati, potranno visualizzare le opere caricate dagli artisti.

Assunzioni:

- L'opera d'arte sarà associata all'artista che l'ha caricata. In questo modo, si potrà partecipare a più concorsi, qualora ce ne fossero, con la stessa opera d'arte.

Requisiti:

Requisito	Descrizione
Caricamento opera d'arte	Un utente registrato, avrà la possibilità di caricare un'opera d'arte compilando il titolo, la descrizione, la tipologia ed inserendo le immagini. Si potranno poi compilare ulteriori campi facoltativi che dipendono dalla tipologia di opera selezionata. Se l'utente possiede almeno un'opera d'arte sarà identificato come artista.
Iscrizione al concorso	Durante la fase di caricamento dell'opera d'arte, sarà possibile iscriverla al concorso. Se l'opzione verrà selezionata, si procederà al pagamento della quota d'iscrizione.
Modifica e cancellazione	L'artista proprietario dell'opera d'arte potrà decidere di modificare le informazioni o di eliminarla. L'opzione sarà disponibile nel suo profilo.
Lista delle opere d'arte	I visitatori del sito, a prescindere che siano registrati o meno, potranno accedere alla lista completa di tutte le opere d'arte. Per ogni opera saranno visualizzate l'immagine, il titolo e l'artista. Infine, ci sarà la possibilità di filtrare le opere visualizzate.
Visualizzazione opera d'arte	Ogni opera d'arte potrà essere visualizzata nel dettaglio da tutti i visitatori del sito. Nella pagina relativa all'opera, saranno visualizzate tutte le informazioni e le immagini associate. Sarà, inoltre, possibile visualizzare tutti i concorsi a cui è iscritta e la relativa posizione in classifica.
Barra di ricerca	Nella parte superiore della piattaforma, sarà presente una barra di ricerca per cercare le opere d'arte.
Segnalazione	Ci sarà la possibilità di segnalare un'opera d'arte che non rispetta le linee guida della piattaforma o che risulta essere offensiva.

Tabella 4.3: AdR: Opere d'arte

4.1.4 Artisti

Obiettivo: Dare la possibilità ai visitatori del sito di visualizzare gli artisti iscritti alla piattaforma, il loro profilo, le loro opere d'arte e i concorsi a cui sono iscritti.

Descrizione: Ogni utente che visita il sito avrà la possibilità di visualizzare tutti gli artisti iscritti alla piattaforma. Per ogni artista sarà possibile accedere al suo profilo e visualizzare le relative informazioni, le opere d'arte caricate ed i concorsi a cui è iscritto.

Assunzioni: Nessuna assunzione.

Requisiti:

Requisito	Descrizione
Lista degli artisti	Sarà possibile visualizzare la lista di tutti gli artisti iscritti alla piattaforma. Sarà inoltre possibile fare una ricerca tra gli artisti.
Visualizzazione	Sarà possibile visualizzare il profilo dell'artista dove saranno disponibili la sua Biografia, le opere d'arte caricate ed i concorsi a cui è iscritto.
Segnalazione	Ci sarà la possibilità di segnalare un artista che non rispetta le linee guida della piattaforma.

Tabella 4.4: AdR: Artisti

4.1.5 Concorsi

Obiettivo: Dare la possibilità di visualizzare i concorsi e le opere d'arte iscritte. Dare la possibilità agli artisti di iscrivere ad un concorso una o più opere d'arte.

Descrizione: Sarà possibile visualizzare tutti i concorsi che la piattaforma organizza. Per ogni concorso sarà disponibile la lista delle opere d'arte e degli artisti iscritti. Se il concorso è in fase di svolgimento o terminato, sarà disponibile anche la classifica. Ogni artista avrà la possibilità di iscrivere una o più opere d'arte ad ogni concorso.

Assunzioni: Nessuna assunzione.

Requisiti:

Requisito	Descrizione
Lista concorsi	Sarà possibile visionare la lista dei concorsi organizzati dalla piattaforma. Per ogni concorso sarà possibile visualizzare le seguenti informazioni: nome, data di inizio iscrizioni, data di fine iscrizioni, data di inizio concorso e data di fine concorso. I concorsi saranno visualizzati in ordine cronologico. Sarà possibile filtrarli e cercarli.
Visualizzazione	Oltre alle informazioni già presenti nella lista generale, sarà possibile visualizzare l'elenco degli artisti e delle opere d'arte iscritte al concorso. Se il concorso è in fase di svolgimento o terminato, sarà disponibile anche la classifica.
Iscrizione al concorso	Ci sarà la possibilità di iscriversi al concorso con una o più opere d'arte già caricate nella piattaforma. Una volta selezionate le opere d'arte da iscrivere, si verrà reindirizzati al pagamento della quota d'iscrizione.

Tabella 4.5: AdR: Concorsi

4.1.6 Profilo e Impostazioni

Obiettivo: Dare la possibilità ad un utente di gestire le proprie informazioni e preferenze.

Descrizione: Ogni utente registrato alla piattaforma avrà una sua area personale dove poter gestire le proprie informazioni e le impostazioni del profilo. In particolare, avrà la possibilità di modificare le sue informazioni personali e la sua biografia, gestire le opere d'arte caricate ed accedere alla sezione delle impostazioni. Dalle impostazioni potrà gestire la MFA, la lingua della piattaforma ed il *light/dark mode*.

Assunzioni: Nessuna assunzione.

Requisiti:

Requisito	Descrizione
Visualizzazione profilo	Ogni utente avrà la possibilità di visualizzare il proprio profilo con le relative informazioni, le opere d'arte caricate ed i concorsi a cui è iscritto. Per ogni opera d'arte avrà la possibilità di modificarla o eliminarla.
Modifica profilo	L'utente sarà in grado di modificare le sue informazioni personali, la foto profilo e la biografia. In particolare, la biografia potrà modificarla attraverso un editor WYSIWYG (<i>What You See Is What You Get</i>).
Impostazioni - Cambio password	Dalle impostazioni, l'utente avrà la possibilità di cambiare la propria password inserendo quella attuale e quella nuova.
Gestione della MFA	Ci sarà la possibilità di abilitare o disabilitare la MFA.
Gestione lingua	Dalle impostazioni, l'utente avrà la possibilità di selezionare la lingua di visualizzazione della piattaforma.
Light/dark mode	Dalle impostazioni, l'utente avrà la possibilità di selezionare la modalità light o dark, in base alla sua preferenza.

Tabella 4.6: AdR: Profilo e Impostazioni

4.1.7 Home Page

Obiettivo: Creare una home page dinamica, accattivante e completa di informazioni ed elementi relativi ai concorsi, agli artisti ed alle opere d'arte.

Descrizione: La Home Page è la pagina di presentazione dell'intero progetto e, come tale, necessita di contenere informazioni complete ed esaustive sul funzionamento della piattaforma e sui partner che sostengono l'iniziativa. Inoltre, dovrà esserci la lista degli artisti iscritti in piattaforma, delle opere d'arte caricate e dei concorsi attivi.

Assunzioni: Nessuna assunzione.

Requisiti:

Requisito	Descrizione
Sezione informativa	Nel contenuto <i>above the fold</i> della home page ci sarà una breve descrizione della piattaforma, delle sue funzionalità e del suo scopo, con la possibilità di guardare un video introduttivo. Come <i>call to action</i> sarà presente un pulsante che rimanda alla sezione di caricamento dell'opera d'arte.
Barra di ricerca	Nella parte alta sarà presente una barra di ricerca con la quale sarà possibile cercare opere d'arte e artisti all'interno della piattaforma.
Opere d'arte	Sarà visibile una lista finita delle opere d'arte più votate all'interno della piattaforma ed una lista delle ultime opere d'arte caricate dagli artisti. Poi un pulsante per accedere alla sezione con tutte le opere d'arte disponibili.
Concorsi	Saranno visibili delle sezioni dedicate a presentare i concorsi attivi nella piattaforma a cui un artista può iscriversi e un utente può votare. Ogni sezione avrà un pulsante che rimanda alla pagina del concorso.
Artisti	Sarà visibile una lista finita degli artisti più votati all'interno della piattaforma ed una lista degli ultimi artisti iscritti. Poi un pulsante per accedere alla sezione con tutti gli artisti presenti.
Partner	Al fondo della home page, sarà presente un carosello con i loghi dei partner che aderiscono al progetto.

Tabella 4.7: AdR: Home Page

4.1.8 Menù

Obiettivo: Fornire un menù intuitivo che contenga le sezioni più importanti.

Descrizione: Il menù dovrà avere il logo della piattaforma ed i link alle seguenti aree: Concorsi, Artisti, Opere d'arte e Login/Registrazione. Nel caso l'utente abbia già fatto login, il link di Login/Registrazione verrà sostituito dal link al profilo. Il menù dovrà essere responsive e nei dispositivi mobile dovrà apparire l'icona hamburger per poterlo visualizzare.

Assunzioni: Nessuna assunzione.

Requisiti:

Requisito	Descrizione
Logo	Sarà presente il logo della piattaforma in alto a sinistra. Cliccando, si verrà rimandati alla home page.
Concorsi	Link che rimanda alla pagina dei concorsi.
Artisti	Link che rimanda alla pagina degli artisti.
Opere d'arte	Link che rimanda alla pagina delle opere d'arte.
Login/Registrazione o Profilo	Sarà presente un pulsante ben visibile per fare il login o la registrazione alla piattaforma. Se l'utente ha già effettuato l'accesso, sarà presente il link che rimanda al suo profilo.

Tabella 4.8: AdR: Menù

4.1.9 Votazione

Obiettivo: Dare la possibilità agli utenti della piattaforma di votare un'opera d'arte all'interno di un concorso.

Descrizione: Durante lo svolgimento di un concorso, sarà possibile votare le opere d'arte iscritte per poter contribuire alla formazione della classifica finale. Ogni utente potrà votare un'opera d'arte, iscritta ad uno specifico concorso, soltanto una volta. Durante la fase di votazione, l'utente potrà assegnare una valutazione che va da 1 a 5 punti e potrà anche lasciare un commento. Il voto dato dall'utente verrà finalizzato con il pagamento di una quota, proporzionale ai punti assegnati. Il punteggio finale di un'opera d'arte, sarà dato dalla sommatoria di tutti i punti ricevuti nei voti.

Assunzioni:

- Solo un utente registrato può votare un'opera d'arte;
- Un artista può votare le opere d'arte, ma non le proprie.

Requisiti:

Requisito	Descrizione
Votazione	Sarà possibile votare un'opera d'arte all'interno di un concorso al quale è iscritta assegnando un punteggio che va da 1 a 5 e lasciando un commento non obbligatorio. Per finalizzare la votazione, sarà necessario procedere al pagamento di una quota proporzionale al punteggio assegnato.

Tabella 4.9: AdR: Votazione

4.1.10 Dashboard di amministrazione

Obiettivo: Dare una visione generale sugli aspetti fondamentali della piattaforma attraverso metriche e statistiche rilevanti. Inoltre, deve dare la possibilità al *Customer Care* di gestire le richieste di assistenza visualizzando e modificando i dati degli utenti, dei concorsi e delle opere d'arte.

Descrizione: Gli utenti con ruolo di amministratore potranno accedere alla piattaforma di amministrazione in cui avranno la possibilità di gestire gli utenti iscritti, gli artisti, le opere d'arte caricate ed i concorsi attivi. Inoltre, visualizzare i dati relativi alle iscrizioni, alle votazioni ed ai pagamenti effettuati. Queste funzionalità saranno necessarie al Customer Care per gestire al meglio ogni richiesta di assistenza. Saranno infine disponibili report, grafici e metriche.

Assunzioni: Nessuna assunzione.

Requisiti:

Requisito	Descrizione
Dashboard	Nella dashboard saranno disponibili diverse metriche, quali numero di utenti iscritti, numero di artisti, numero di opere d'arte caricate, iscrizioni effettuate, voti finalizzati e pagamenti elaborati.
Utenti	Ci sarà la possibilità di gestire gli utenti iscritti alla piattaforma. Per ogni utente sarà possibile gestire le informazioni quali biografia, dati anagrafici, opere d'arte collegate, iscrizioni e votazioni.
Concorsi	Sarà possibile gestire i concorsi della piattaforma. Per ogni concorso sarà possibile gestire le informazioni e visualizzare le metriche quali iscrizioni, opere d'arte, classifica e voti.
Opere d'arte	Ci sarà la possibilità di gestire le opere d'arte caricate nella piattaforma e le relative iscrizioni e votazioni.

Tabella 4.10: AdR: Dashboard di amministrazione

4.2 Progettazione

Una volta completata la fase di analisi dei requisiti ed essersi confrontati con il cliente per ricevere l'approvazione, si può procedere con la fase di progettazione dell'applicativo. Questa fase è la più importante e dev'esserci dedicata la giusta attenzione perché dipenderanno costi e performance dell'intero progetto. In questa sezione si analizzeranno il modello dei dati ed il design del database, l'architettura dell'applicativo ed i servizi coinvolti, i flussi di utilizzo della piattaforma e la previsione di costo dell'intera infrastruttura.

4.2.1 Modello dei dati

Dall'analisi dei requisiti, sono emerse 5 entità: USER, ARTWORK, CONTEST, SUBSCRIPTION e VOTE. Si procederà all'analisi di ogni entità identificando i dati che la compongono e la dimensione media di ogni elemento. Inoltre, si andranno ad individuare le possibili query che verranno utilizzate dall'applicativo. Queste saranno molto utili durante la fase di creazione del database per strutturare gli indici.

USER

DATI	TIPOLOGIA	DIMENSIONE
id	UUID	16 Byte
name	String	20 Byte
surname	String	20 Byte
username	String	20 Byte
description	String	1 KByte
email	String	30 Byte
fiscal_code	String	16 Byte
gender	String	6 Byte
profile_image	String	100 Byte
is_artist	Boolean	1 bit
facebook	String	100 Byte
instagram	String	100 Byte
linkedin	String	100 Byte
is_dark_mode	Boolean	1 bit
language	String	5 Byte
timestamp	Number	4 Byte

Tabella 4.11: Modello dei dati: USER

Ogni record USER peserà, mediamente, 1.5 KByte

Le query identificate per l'entità USER sono:

- Lista degli utenti.
- Lista degli utenti classificati come artisti.
- Dettagli di un utente.

ARTWORK

DATI	TIPOLOGIA	DIMENSIONE
id	UUID	16 Byte
user_id	UUID	16 Byte
title	String	100 Byte
description	String	1 KByte
year	Number	4 Byte
images	[String]	100 Byte / immagine
status	String	20 Byte
type	String	20 Byte
rarity	String	15 Byte
width	Number	4 Byte
height	Number	100 Byte
depth	Number	100 Byte
weight	Number	100 Byte
materials	[String]	100 Byte
painting_category	[String]	100 Byte
painting_style	[String]	100 Byte
painting_techniques	[String]	100 Byte
photographic_themes	[String]	100 Byte
photographic_currents	[String]	100 Byte
colors	[String]	100 Byte
tags	[String]	100 Byte
timestamp	Number	4 Byte

Tabella 4.12: Modello dei dati: ARTWORK

Ogni record ARTWORK peserà, mediamente, 2.5 KByte

Le query identificate per l'entità ARTWORK sono:

- Lista delle opere d'arte.
- Lista delle opere d'arte di un utente.
- Dettagli di un'opera d'arte.

CONTEST

DATI	TIPOLOGIA	DIMENSIONE
id	UUID	16 Byte
title	String	100 Byte
description	String	1 KByte
image	String	100 Byte
subscription_date_start	Date	4 Byte
subscription_date_end	Date	4 Byte
contest_date_start	Date	4 Byte
contest_date_end	Date	4 Byte
price	Number	32 Byte
stripe_price_id	String	50 Byte
additional_price	Number	32 Byte
stripe_additional_price_id	String	50 Byte
timestamp	Number	4 Byte

Tabella 4.13: Modello dei dati: CONTEST

Ogni record CONTEST peserà, mediamente, 1.5 KByte

Le query identificate per l'entità CONTEST sono:

- Lista dei contest.
- Dettagli di un contest.

SUBSCRIPTION

DATI	TIPOLOGIA	DIMENSIONE
id	UUID	16 Byte
contest_id	UUID	16 Byte
artwork_id	UUID	16 Byte
user_id	UUID	16 Byte
payment_id	String	16 Byte
ranking_votes	Number	4 Byte
timestamp	Number	4 Byte

Tabella 4.14: Modello dei dati: SUBSCRIPTION

Ogni record SUBSCRIPTION peserà, mediamente, 88 Byte

Le query identificate per l'entità SUBSCRIPTION sono:

- Lista delle opere iscritte ad un concorso.
- Lista dei concorsi a cui è iscritta un'opera.
- Lista degli artisti partecipanti ad un concorso.
- Dettagli di un'iscrizione.
- Classifica di un concorso.

VOTE

id	UUID	16 Byte
contest_id	UUID	16 Byte
artwork_id	UUID	16 Byte
user_id	UUID	16 Byte
value	Number	4 Byte
comment	String	1 KByte
payment_id	String	16 Byte
timestamp	Number	4 Byte

Tabella 4.15: Modello dei dati: VOTE

Ogni record VOTE peserà, mediamente, 1 KByte

Le query identificate per l'entità VOTE sono:

- Lista dei voti ricevuti da un'opera.
- Lista dei voti ricevuti da un'opera durante un concorso.

4.2.2 DynamoDB - Single Table Design

Per la gestione dei dati dell'applicazione si utilizzerà il servizio PaaS *Amazon DynamoDB*, il database serverless non relazionale (*NoSQL*) del tipo chiave/valore (*Key/Value*) fornito da AWS. Verrà adottato l'approccio *Single Table Design*, best practice dell'application design di AWS. Questa metodologia consiste nell'avere tutti i dati dell'applicativo nella stessa tabella. [21] [24] La *Partition Key* può essere pensata come la chiave primaria del database e viene identificata dal campo *id*. Il campo *id* è del tipo UUID e viene generato automaticamente. La *Sort Key*, utilizzata per identificare la tipologia del record, è il campo *type*. Il campo *type* è del tipo String e può assumere i seguenti valori: USER, ARTWORK, CONTEST, SUBSCRIPTION, VOTE. Di seguito verranno presentati degli esempi di record per ogni tipologia di dato.

USER

```
1 {
2   "id": "1e057312-6a2f-48bc-98b8-efd6393ddab6",
3   "type": "USER",
4   "name": "Federico",
5   "surname": "Masiero",
6   "username": "federicomasiero",
7   "description": "<p>La mia Biografia!</p>",
8   "email": "federico.masiero@sveloop.com",
9   "fiscal_code": "MSRFRC95B21G224F",
10  "gender": "MALE",
11  "profile_image": "/1e057312-6a2f-48bc-98b8-efd6393ddab6
    /1664607337/profile.png",
12  "is_artist": FALSE,
13  "is_dark_mode": TRUE,
14  "language": "it-IT",
15  "timestamp": 1664607437
16 }
```

Codice 4.1: DynamoDB Element - USER

ARTWORK

```
1 {
2   "id": "e5b0f596-3802-4b8d-aec5-e153b305f0f0",
3   "user_id": "1e057312-6a2f-48bc-98b8-efd6393ddab6",
4   "title": "La mia prima opera d'arte!",
5   "description": "<p>La mia prima opera d'arte!</p>",
6   "year": 2022,
7   "images": [
8     "/e5b0f596-3802-4b8d-aec5-e153b305f0f0/1664607785/artwork1.png",
9     "/e5b0f596-3802-4b8d-aec5-e153b305f0f0/1664607808/artwork2.png",
10    ],
11   "status": "AVAILABLE",
12   "type": "PAINTING",
13   ...
14   "timestamp": 1664607854
15 }
```

Codice 4.2: DynamoDB Element - ARTWORK

CONTEST

```
1 {
2   "id": "818ce4d0-daf6-437b-b74f-a820910f8472",
3   "title": "Contest 111",
4   "description": "<p>Benvenuti a Contest 111</p>",
5   "image": "/818ce4d0-daf6-437b-b74f-a820910f8472/1655893693/
6     contest111.png",
7   "subscription_date_start": "2022-07-25T00:00:01+02",
8   "subscription_date_end": "2022-10-19T23:59:59+02",
9   "contest_date_start": "2022-10-20T00:00:01+02",
10  "contest_date_end": "2023-02-07T23:59:59+02",
11  "price": 111,
12  "stripe_price_id": "price_1LEU6XJlC0CigirWMGWi4pqw",
13  "additional_price": 9,
14  "stripe_additional_price_id": "price_1LEWSmJlC0CigirWljpW7qnk",
15  "timestamp": 1655893693,
}
```

Codice 4.3: DynamoDB Element - CONTEST

SUBSCRIPTION

```
1 {  
2   "id": "f534998c-6f2c-45cd-9c0d-7a8f83354ded",  
3   "contest_id": "818ce4d0-daf6-437b-b74f-a820910f8472",  
4   "artwork_id": "e5b0f596-3802-4b8d-aec5-e153b305f0f0",  
5   "user_id": "1e057312-6a2f-48bc-98b8-efd6393ddab6",  
6   "payment_id": "pi_3LlwQaCgkj5wc0G1023YgxzD",  
7   "ranking_votes": 0,  
8   "timestamp": 1664608711,  
9 }
```

Codice 4.4: DynamoDB Element - SUBSCRIPTION

VOTE

```
1 {
2   "id": "94937def-b50a-4c27-83d8-b88e397621fb",
3   "contest_id": "818ce4d0-daf6-437b-b74f-a820910f8472",
4   "artwork_id": "e5b0f596-3802-4b8d-aec5-e153b305f0f0",
5   "user_id": "1e057312-6a2f-48bc-98b8-efd6393ddab6",
6   "value": 5,
7   "comment": "<p>Bella opera d'arte!</p>",
8   "payment_id": "pi_3LPXoJCgkj5wc0G110VdW2wv",
9   "timestamp": 1664608961,
10 }
```

Codice 4.5: DynamoDB Element - VOTE

Global Secondary Index

In seguito all'analisi delle possibili query che verranno fatte nel database, si è deciso di introdurre i seguenti *Global Secondary Index* (GSI) per ottimizzare tempo e risorse.

GSI	Partition Key	Sort Key
artwork-type-index	artwork_id	type
contest-type-index	contest_id	type
contest_id-ranking_votes-index	contest_id	ranking_votes
payment_id-index	payment_id	—
type-index	type	—
user-type-index	user_id	type

Tabella 4.16: DynamoDB - Global Secondary Index

4.2.3 Architettura dell'applicativo

In questa sezione viene presentata l'architettura dell'applicativo e le modalità di utilizzo dei servizi scelti. L'intera applicazione è stata progettata per avere un'architettura basata sugli eventi con l'utilizzo di servizi serverless PaaS forniti da AWS. Di seguito viene presentato lo schema dell'architettura con i servizi utilizzati e le possibili interazioni tra di loro.

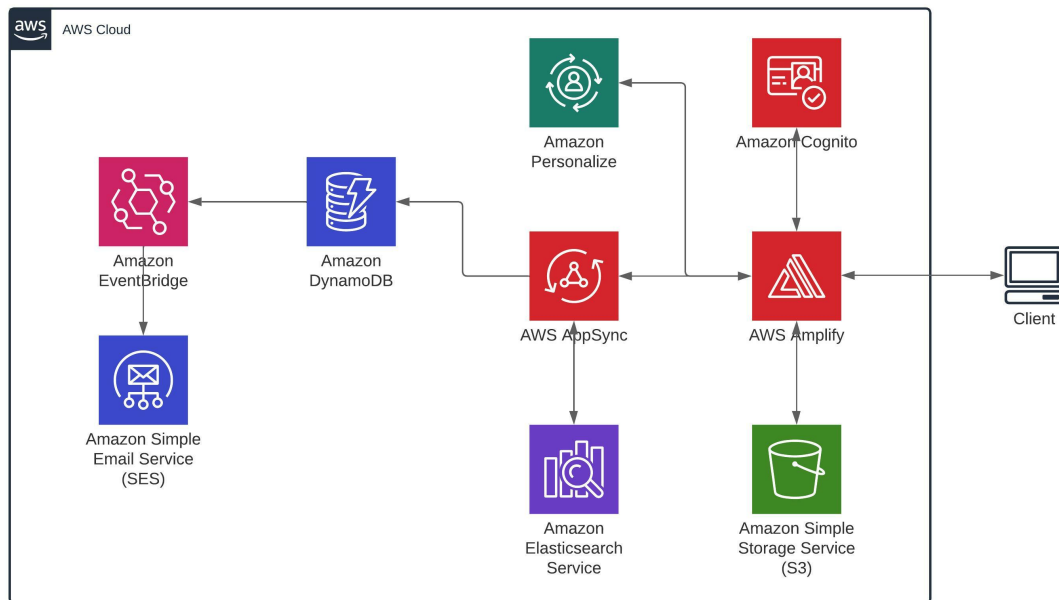


Figura 4.1: Architettura Software Contest111

L'applicazione viene ospitata e distribuita dal servizio di hosting server *ASW Amplify*. Per gestire l'accesso e la registrazione degli utenti, l'applicazione potrà comunicare direttamente con il servizio *Amazon Cognito*. Per la gestione dei file multimediali, quali immagini e video, l'applicativo potrà interfacciarsi direttamente con il servizio *Amazon S3*. Per ricevere suggerimenti, basati sui dati, in merito agli artisti ed alle opere d'arte da mostrare ad ogni utente, si potrà interrogare il servizio *Amazon Personalize*. Per accedere a qualsiasi tipo di dato come opere d'arte, artisti, concorsi, iscrizioni, classifiche o votazioni, l'applicazione dovrà interfacciarsi con il servizio di *API GraphQL AWS AppSync*. Questo si occuperà di ricevere la richiesta, di elaborarla interrogando i servizi necessari e di restituire solo i dati richiesti. In particolare *AWS AppSync* si potrà interfacciare con il servizio di database NoSQL *Amazon DynamoDB* ed il servizio di ricerca *Amazon OpenSearch*. Infine, il servizio di gestione degli eventi *Amazon EventBridge* si occuperà di inviare delle notifiche attraverso i servizi *Amazon SES* e/o *Amazon SNS* ogni qualvolta che una specifica azione si verifica (e.g. nuovo record inserito nel database).

4.2.4 Flussi di utilizzo dell'applicazione

In questa sezione vengono approfonditi i flussi per l'utilizzo della piattaforma. Vengono descritti i collegamenti tra le varie aree e le possibili azioni che possono intraprendere gli utenti, con le relative condizioni che devono essere soddisfatte.

Flusso Generale

Di seguito, lo schema che rappresenta il flusso generale dell'intera piattaforma.

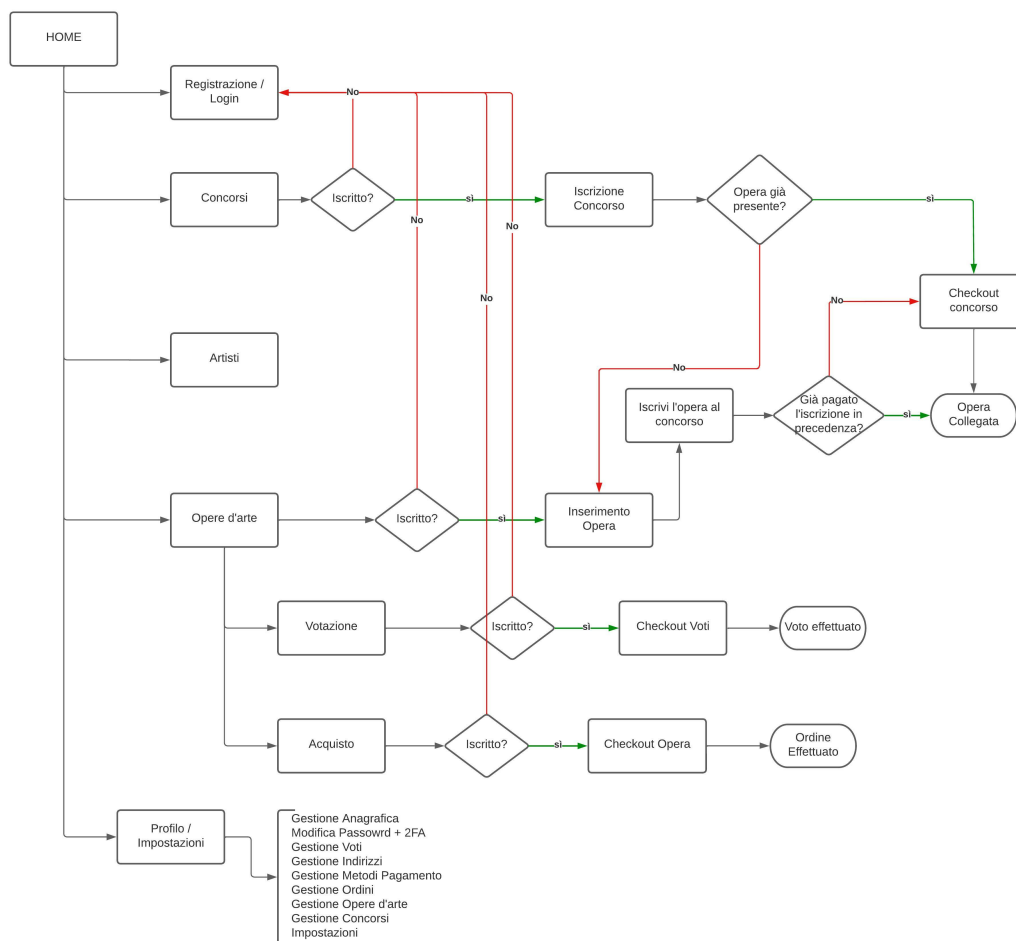


Figura 4.2: Flusso Generale

Flusso Login e Registrazione

Di seguito, lo schema che rappresenta il flusso di registrazione, login, autenticazione a due fattori e attivazione dell'account.

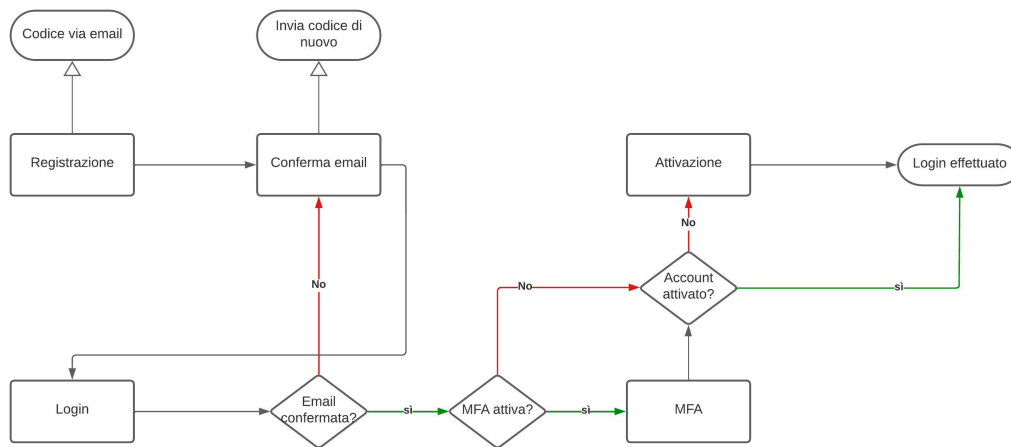


Figura 4.3: Flusso Login e Registrazione

Flusso caricamento opere d'arte e iscrizione concorsi

Di seguito, lo schema che rappresenta il flusso per il caricamento di una o più opere d'arte e l'iscrizione ai concorsi.

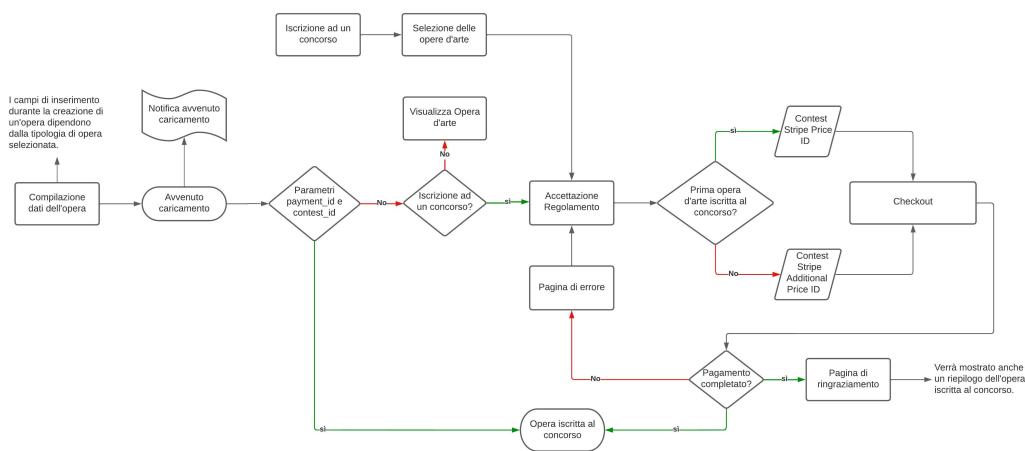


Figura 4.4: Flusso caricamento opere d'arte e iscrizione concorsi

Flusso per la votazione

Di seguito, lo schema che rappresenta il flusso per la votazione di un'opera.

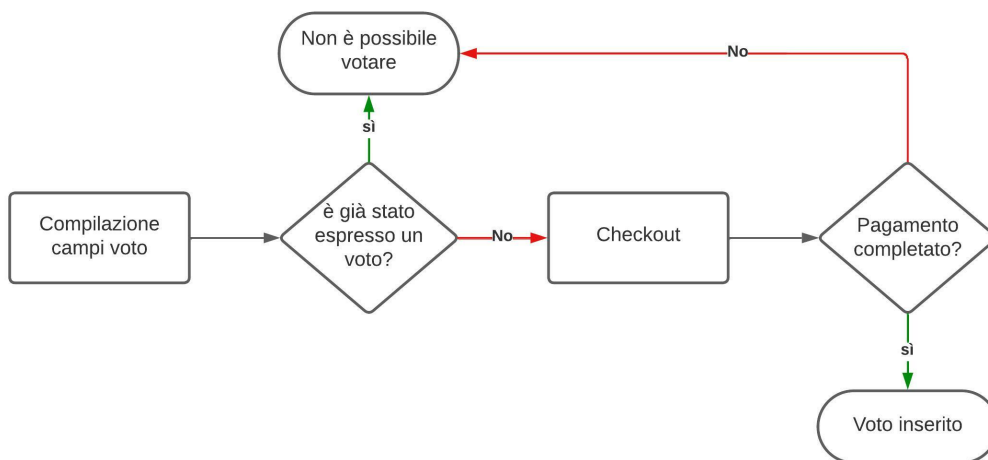


Figura 4.5: Flusso per la votazione

4.2.5 Previsione di costo del cloud

Terminata l'analisi dei requisiti, svolti diversi meeting con il cliente e con il team marketing e conclusa la fase di progettazione, si è potuto stimare i seguenti parametri. Partendo da questi, si è potuto ricavare la previsione di costo dell'intera infrastruttura cloud.

Utenti registrati / mese	1.000
% Utenti attivi / mese	50%
% Utenti Artisti	10%
Regioni extra coinvolte	3
Numero medio opere / artista	5
Numero medio update opera / artista / mese	2
Numero medio read / mese / utente	1.000
Numero medio update informazioni / artista / mese	10
Peso medio artwork document (KB)	10
Numero medio iscrizioni contest / opera d'arte	1
Numero medio di voti / opera d'arte	10
Numero medio immagini / opera d'arte	10
Peso medio immagine dell'opera d'arte (MB)	5
Dimensione pagina media (MB)	1,5
Suggerimenti medi / ora / utente	2
Numero suggeritori utilizzati	1
Ore al mese	732

Tabella 4.17: Parametri per la stima di costo del cloud

Di seguito, la previsione di costo dell'infrastruttura cloud.

Servizio / Situazione	Costo unitario	luglio 2022	agosto 2022	settembre 2022	ottobre 2022	novembre 2022	dicembre 2022	gennaio 2023	febbraio 2023
Utenti registrati		0,00	1.000,00	2.000,00	3.000,00	4.000,00	5.000,00	6.000,00	7.000,00
Utenti attivi		0,00	1.000,00	1.500,00	2.000,00	2.500,00	3.000,00	3.500,00	4.000,00
Richieste Write		0,00	6.000,00	7.000,00	8.000,00	9.000,00	10.000,00	11.000,00	12.000,00
Richieste Read		0,00	1.000.000,00	1.500.000,00	2.000.000,00	2.500.000,00	3.000.000,00	3.500.000,00	4.000.000,00
Dimensione Tabella (GB)		0,00	0,07	0,13	0,20	0,26	0,33	0,39	0,46
Dimensione Storage		0,00	25,00	50,00	75,00	100,00	125,00	150,00	175,00
Numero di opere presenti		0,00	500,00	1.000,00	1.500,00	2.000,00	2.500,00	3.000,00	3.500,00
TOTALI		€ 0,00	€ 74,97	€ 82,00	€ 89,03	€ 96,15	€ 103,26	€ 110,38	€ 117,49
Cognito	0,0055	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00
DynamoDB	1,525	€ 0,00	€ 1,53	€ 2,30	€ 3,06	€ 3,83	€ 4,59	€ 5,35	€ 6,12
DynamoDB Storage	0,306	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00
DynamoDB Global Table	2,288	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,00
DynamoDB Backup	0,2448	€ 0,00	€ 0,02	€ 0,03	€ 0,05	€ 0,06	€ 0,08	€ 0,10	€ 0,11
S3 Storage	0,0245	€ 0,00	€ 0,61	€ 1,23	€ 1,84	€ 2,45	€ 3,06	€ 3,68	€ 4,29
S3 PUT Request	0,0054	€ 0,00	€ 0,03	€ 0,03	€ 0,03	€ 0,03	€ 0,03	€ 0,03	€ 0,03
S3 GET Request	0,00043	€ 0,00	€ 4,30	€ 6,45	€ 8,60	€ 10,75	€ 12,90	€ 15,05	€ 17,20
AppSync	4	€ 0,00	€ 4,02	€ 6,03	€ 8,03	€ 10,04	€ 12,04	€ 14,04	€ 16,05
Amplify	0,15	€ 0,00	€ 0,23	€ 0,34	€ 0,45	€ 0,56	€ 0,68	€ 0,79	€ 0,90
Open Search	0,084	€ 0,00	€ 61,49	€ 61,49	€ 61,49	€ 61,49	€ 61,49	€ 61,49	€ 61,49
Personalize - Utente		€ 0,00	€ 2,75	€ 4,12	€ 5,49	€ 6,86	€ 8,24	€ 9,61	€ 10,98
Personalize - Suggerimenti	0,0833	€ 0,00	€ 0,00	€ 0,00	€ 0,00	€ 0,08	€ 0,17	€ 0,25	€ 0,33

Capitolo 5

Sviluppo

In questo capitolo verranno analizzate le varie fasi di sviluppo del progetto, dal *Project Management* fino al *deploy* nell'ambiente di *production*. Per ogni sezione verrà descritto cosa è stato fatto, illustrando alcuni esempi di codice, e come sono state affrontate le sfide che sono emerse durante la realizzazione del progetto. La *AWS Region* scelta per la produzione è *eu-central-1* (Francoforte).

5.1 Design mockup

Affinché il progetto potesse avere una UI/UX intuitiva e che prestasse particolare attenzione al design, si è deciso di creare dei mockup interattivi dell'applicativo (Appendice A). Per fare ciò il designer del team di sviluppo si è avvalso dello strumento Figma [10] che permette di disegnare il *front end* di un'applicazione e renderlo interattivo così da poter essere provato simulando un ambiente reale. Questa scelta si è rivelata decisiva per la buona riuscita del progetto in quanto il business developer e il committente hanno potuto dare i loro feedback in modo mirato prima che la fase di codifica iniziasse. Infine, lo sviluppatore che si è occupato della realizzazione del *front end*, ha potuto usufruire di una “guida” grafica per l'implementazione.

5.2 Project Management

La realizzazione di questo progetto ha coinvolto molte persone provenienti da diverse parti dell'Italia che hanno lavorato esclusivamente da remoto. Per dirigere il team di sviluppo e fornire le indicazioni necessarie alla corretta implementazione dell'applicativo, ricevere i feedback da parte degli *stakeholders* e controllare gli avanzamenti per avere il progetto sempre sotto controllo, si è deciso di adottare la Metodologia Agile [5] e nello specifico il *framework Scrum*. [7] Per la fase di *Bug Fixing* e *Beta Testing*, invece, si è deciso di adottare un approccio differente: il *framework Kanban*. [6]

Terminate le fasi di analisi dei requisiti e di design, si è redatta la *roadmap* dell'intero progetto, con lo scopo di dettare i tempi per l'implementazione delle varie funzionalità. La tabella di marcia prevedeva l'inizio dello sviluppo nella prima settimana di Maggio e la fine dei lavori di implementazione nell'ultima settimana di Giugno. Il mese di Luglio sarebbe stato poi utilizzato per le fasi di *Bug Fixing* e *Beta Testing*, prima del rilascio al pubblico previsto per la fine del mese di Luglio.

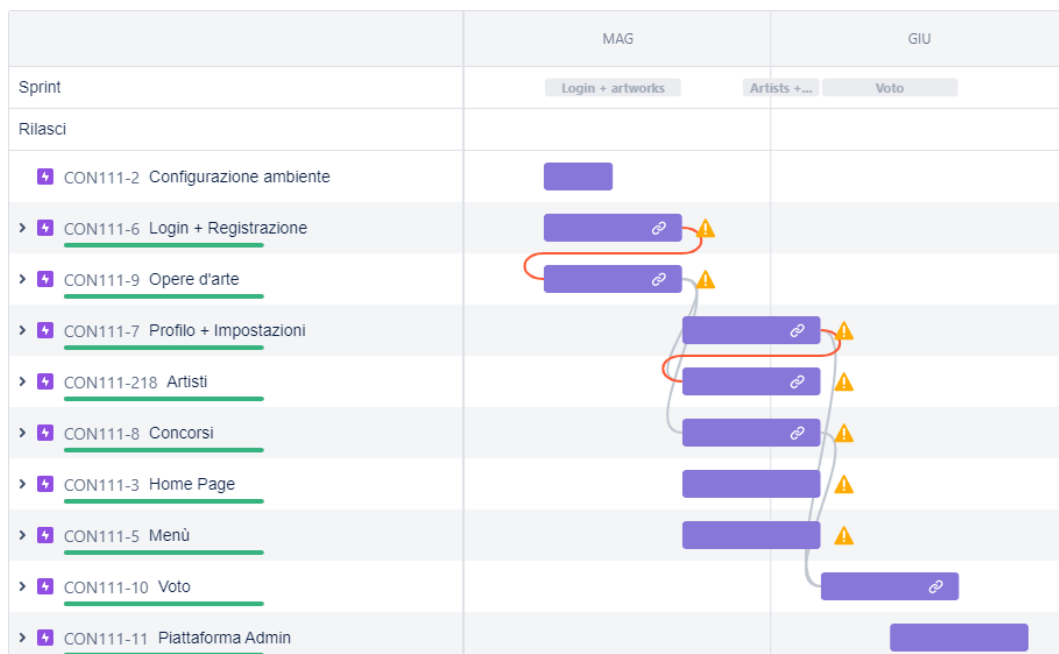


Figura 5.1: Roadmap del progetto Contest111

Nel *framework Scrum*, gli *Sprint* sono un insieme di funzionalità da completare nell'arco di un tempo specifico, solitamente una o due settimane lavorative. Per questo progetto si è deciso di adottare *Sprint* della durata di due settimane lavorative. Di conseguenza, la *roadmap* è stata suddivisa in quattro *Sprint*:

- Login + Opere d'arte
- Artisti + Concorsi

- Votazioni
- Dashboard di amministrazione

Ogni mattina alle ore 09:00, veniva effettuato un meeting su Zoom della durata di massimo 15 minuti. Questo meeting giornaliero, nel framework Scrum, viene definito “*StandUp*” ed ha come principali scopi fare Team Building e aggiornarsi sugli ultimi sviluppi del progetto. Al termine di ogni Sprint, è stato effettuato un meeting “*Retrospectives*” che aveva lo scopo di fare il punto della situazione, capire cosa non fosse andato durante le due settimane precedenti e trarre degli spunti per migliorare nello Sprint successivo.

5.3 Back End

In questa sezione viene approfondita l'implementazione e la configurazione dei servizi PaaS utilizzati per le operazioni di Back End dell'applicazione. Vengono forniti alcuni codici utilizzati durante l'implementazione.

5.3.1 Amazon Cognito

Per utilizzare le funzionalità di autenticazione fornite dal servizio PaaS *Amazon Cognito*, è necessario creare un Bacino di Utenza (*User Pool*) nella *AWS Region* di produzione. Oltre al classico metodo di autenticazione attraverso email e password, abilitato di default, è possibile attivare il *Federated Identity Provider* che permette di utilizzare le credenziali di Facebook, Google, Amazon e Apple. Durante la fase di configurazione, è necessario scegliere i metodi di accesso consentiti all'utente (email, username o numero di telefono) e le condizioni che ogni password deve rispettare. È necessario scegliere se abilitare l'autenticazione a più fattori e se gestirla attraverso SMS e/o applicazione. Inoltre, si deve scegliere se abilitare la registrazione da parte degli utenti e i dati aggiuntivi da fornire, il metodo di conferma dell'email e del numero telefonico con il rispettivo canale di invio associato (*Amazon SES* o *Amazon SNS*) e come gestire il recupero della password dimenticata. Infine, è necessario configurare le modalità di connessione con l'applicazione ed i relativi permessi. È possibile creare lo *User Pool* e configurarlo in tutte le sue funzionalità direttamente dall'interfaccia di AWS. *Amazon Cognito* può anche essere configurato attraverso il framework serverless *AWS SAM*. Di seguito un codice di esempio dalla documentazione di AWS. [22]

```
1 MyCognitoUserPool:
2   Type: AWS::Cognito::UserPool
3   Properties:
4     UserPoolName: !Ref CognitoUserPoolName
5     Policies:
6       PasswordPolicy:
7         MinimumLength: 8
8     UsernameAttributes:
9       - email
10    Schema:
11      - AttributeDataType: String
12        Name: email
13        Required: false
14
15 MyCognitoUserPoolClient:
16   Type: AWS::Cognito::UserPoolClient
17   Properties:
18     UserPoolId: !Ref MyCognitoUserPool
19     ClientName: !Ref CognitoUserPoolClientName
20     GenerateSecret: false
```

Codice 5.1: Definizione Amazon Cognito in AWS SAM

5.3.2 Amazon S3

Per utilizzare le funzionalità di archiviazione fornite dal servizio PaaS *Amazon S3*, è necessario creare un'istanza contenitore (*Bucket*) ed assegnarci un nome univoco nella *AWS Region*. Dal nome scelto verrà generato l'URL per l'accesso al *bucket*. Durante la fase di creazione si potranno configurare le impostazioni generali di accesso, il versioning e la crittografia degli oggetti caricati. Una volta creata l'istanza di S3, si potranno modificare le policy di accesso. Per il progetto in esame sono state configurate in modo da abilitare l'accesso solo alle richieste HTTP che arrivano dall'applicazione e per le quattro azioni indicate.

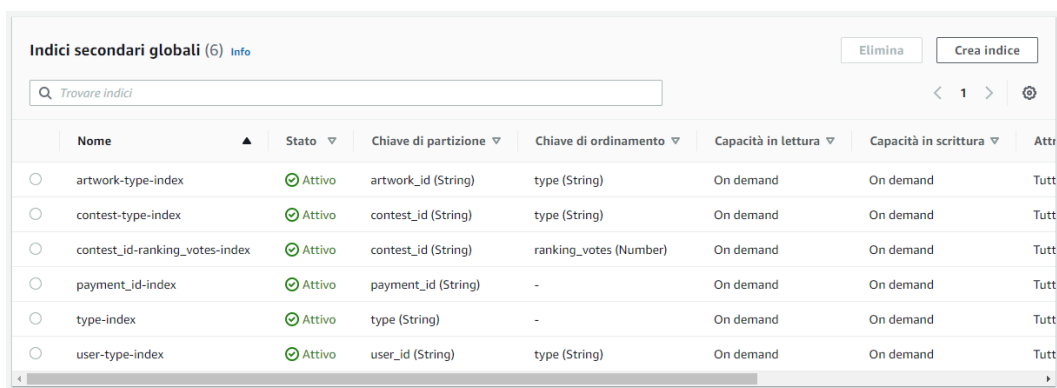
```
1 {
2   "Version": "2012-10-17",
3   "Id": "HTTPRefererPolicy",
4   "Statement": [
5     {
6       "Sid": "AllowRequestsHTTPReferer",
7       "Effect": "Allow",
8       "Principal": "*",
9       "Action": [
10        "s3:GetObject",
11        "s3:GetObjectVersion",
12        "s3:PutObject",
13        "s3:DeleteObject"
14      ],
15       "Resource": "arn:aws:s3:::contest111/*"
16     },
17     {
18       "StringLike": {
19         "aws:Referer": [
20           "https://platform.contest111.com/*"
21         ]
22       }
23     }
24   ]
25 }
```

Codice 5.2: Policy di accesso Amazon S3

È possibile creare il *bucket* e configurarlo in tutte le sue funzionalità direttamente dall'interfaccia di AWS. *Amazon S3* può anche essere configurato attraverso il framework serverless *AWS SAM*.

5.3.3 Amazon DynamoDB

Per utilizzare le funzionalità fornite dal servizio PaaS *Amazon DynamoDB*, è necessario creare una tabella nella *AWS Region* di produzione. Durante la fase di creazione della tabella è necessario inserire la *Partition Key* e, facoltativamente, la *Sort Key* selezionando la rispettiva tipologia tra String, Number o Binary. Si potranno, poi, configurare le impostazioni della tabella selezionando le capacità di lettura e scrittura oppure l'opzione *On Demand*. Infine, si potranno configurare i *Local Secondary Index* (LSI) che, nel progetto preso in esame, non sono necessari. Una volta creata la tabella, sarà possibile configurare i *Global Secondary Index* (GSI). La configurazione della tabella *DynamoDB* del progetto ha seguito quanto previsto nella fase di progettazione. È stata selezionata l'opzione *On Demand* per le capacità di lettura e scrittura della tabella. Di seguito, l'interfaccia di AWS per la gestione dei GSI nella tabella del progetto.



	Nome	Stato	Chiave di partizione	Chiave di ordinamento	Capacità in lettura	Capacità in scrittura	Attr
<input type="radio"/>	artwork-type-index	Attivo	artwork_id (String)	type (String)	On demand	On demand	Tutt
<input type="radio"/>	contest-type-index	Attivo	contest_id (String)	type (String)	On demand	On demand	Tutt
<input type="radio"/>	contest_id-ranking_votes-index	Attivo	contest_id (String)	ranking_votes (Number)	On demand	On demand	Tutt
<input type="radio"/>	payment_id-index	Attivo	payment_id (String)	-	On demand	On demand	Tutt
<input type="radio"/>	type-index	Attivo	type (String)	-	On demand	On demand	Tutt
<input type="radio"/>	user-type-index	Attivo	user_id (String)	type (String)	On demand	On demand	Tutt

Figura 5.2: Sviluppo: DynamoDB - Global Secondary Index

È possibile creare la tabella e configurarla in tutte le sue funzionalità direttamente dall'interfaccia di AWS. *Amazon DynamoDB* può anche essere configurato attraverso il framework serverless *AWS SAM*.

5.3.4 AWS AppSync

Per utilizzare le funzionalità fornite dal servizio PaaS *AWS AppSync*, è necessario creare una *API GraphQL* nella *AWS Region* di produzione. Durante la fase di creazione si dovrà indicare un nome che verrà utilizzato internamente. Una volta creata la nuova API si otterrà l'URL a cui l'applicativo potrà inviare le richieste GraphQL e si potranno configurare le modalità di accesso ai dati. Il progetto preso in esame utilizza due modalità: *API Key* e *Cognito User Pool*. Questa duplice modalità si rende necessaria per assicurare una corretta logica nella gestione delle chiamate (e.g. solo un utente registrato potrà inserire un'opera d'arte).

Come primo passaggio nella configurazione dell'API, sarà necessario definire lo schema GraphQL. Nell'appendice B è presente un estratto dello schema dell'applicazione presa in esame.

AppSync si occupa di interrogare, attraverso i *resolver*, le diverse origini dei dati per fornire una risposta. Come secondo passaggio di configurazione, si rende necessario il collegamento tra le *query* e le *mutation* dello schema e le origini dei dati. Le tipologie di origini dei dati selezionabili sono:

- Tabelle *Amazon DynamoDB*
- Domini *Amazon OpenSearch*
- Funzioni *AWS Lambda*
- Database Relazionali
- Endpoint HTTP

Ogni *resolver* potrà essere configurato in modo differente in base all'origine dei dati collegata. In particolare, i *resolver* che hanno collegata una tabella *DynamoDB* dovranno essere configurati attraverso una o più funzioni per la mappatura della richiesta nella tabella e per la mappatura della risposta. Queste funzioni, definite con linguaggio *Apache Velocity* (VTL), hanno lo scopo di formattare la query per la tabella e i dati per la risposta. Come ultimo passaggio della fase di configurazione, si può provvedere al collegamento delle *query* e delle *mutation*, definite nello schema, ai rispettivi *resolver*.

Nel progetto in esame è stata collegata un'unica origine dei dati: la tabella *DynamoDB*. Tutte le *query* e le *mutation* definite nello schema GraphQL sono state collegate ai rispettivi *resolver* configurati con specifiche funzioni VTL. Nell'appendice C sono riportati alcuni *resolver* implementati.

È possibile creare l'API *GraphQL* e configurarla in tutte le sue funzionalità direttamente dall'interfaccia di AWS. *AWS AppSync* può anche essere configurato attraverso il framework serverless *AWS SAM*. Infine, l'interfaccia di AWS fornisce la possibilità, eseguendo *query* e *mutation*, di testare direttamente quanto implementato.

Per una questione di necessità, budget e nuovi risvolti nel business model, il committente ha deciso di non procedere con le altre funzionalità previste durante la fase di progettazione. Si potrà pensare, in base a come evolverà il progetto, di implementare quanto già progettato in un futuro rilascio.

5.4 Front End

Per la realizzazione del Front End degli applicativi si è scelto di utilizzare il framework *Next.js* [15] che viene considerato dalla community *React* uno dei più completi e funzionali. In merito alla gestione della grafica, si è scelto di avvalersi del framework *Tailwind CSS*. [29] Per la connessione diretta ai servizi AWS scelti e configurati in Back End, in particolare *Amazon Cognito* ed *Amazon S3*, si è scelto di avvalersi delle *AWS SDK per JavaScript*. [23] Mentre, per l'interrogazione dell'*API GraphQL* si è scelto di avvalersi della libreria *SWR* [33], che viene fornita dallo stesso team di *Next.js*.

Per gestire la newsletter e le email promozionali, il team che si occupa di marketing si è avvalso del servizio *Active Campaign*. [1] Per sincronizzare gli eventi nella piattaforma (e.g. iscrizione alla newsletter) con il software di automazione delle email si è provveduto a collegarli utilizzando le API messe a disposizione da *Active Campaign*.

Infine, si è gestito il *deploy* delle applicazioni avvalendosi del servizio PaaS *AWS Amplify*, che fornisce agli sviluppatori la possibilità di implementare una distribuzione del codice (CI/CD), partendo da una *repository*. Durante la fase di creazione è necessario collegare la *repository git* dell'applicazione, selezionare il ramo principale (solitamente il *branch master*) e configurare le impostazioni della *build*. Una volta creata l'automazione di *deploy*, si potrà procedere al collegamento del dominio personalizzato, alla configurazione delle variabili d'ambiente ed al collegamento di ulteriori rami della repository in sottodomini personalizzati (solitamente il *branch development*). L'applicazione è raggiungibile collegandosi a platform.contest111.com.

5.5 Sviluppi futuri

Lo sviluppo delle applicazioni non termina con il loro rilascio. Oltre a correggere eventuali bug che possono emergere durante il quotidiano utilizzo, si possono implementare nuove funzionalità ed ottimizzare quelle già sviluppate. A patto che il progetto continui a crescere e raccogliere artisti da tutto il mondo, le funzionalità che si prevede di implementare nei futuri rilasci sono:

- **Gamification.** Si vogliono coinvolgere gli utenti della piattaforma con sfide, medaglie e punti per ogni azione che svolgono.
- **Personalizzazione dei risultati.** Attraverso l'utilizzo di *Amazon Personalize* si possono ottenere dei risultati basati sui dati raccolti dalla piattaforma e personalizzati per ogni utente.
- **Marketplace.** Per dare agli artisti la possibilità di monetizzare la loro arte e alla piattaforma un nuovo modello di revenue.

Capitolo 6

Conclusioni

Con questa tesi si è voluto approfondire l'utilizzo delle tecnologie serverless nello sviluppo di applicazioni cloud-native. Si è iniziato con l'approfondire l'argomento dal punto di vista teorico con paper, blog di settore, webinar e documentazioni fornite dai provider. Un approfondito studio teorico delle tecnologie serverless, dello stato dell'arte e delle best practice nello sviluppo di applicazioni cloud-native è stato indispensabile per apprendere fino in fondo le potenzialità di queste tecnologie ed i retroscena del loro funzionamento. In questo modo si è avuto ben chiaro quali sono i vantaggi e le attuali sfide nell'utilizzo del modello di *cloud computing serverless*.

Dall'analisi teorica del panorama attuale in campo serverless è emerso che queste tecnologie sono molto utili ed apprezzabili per progetti in fase di start-up, con picchi di utilizzo degli applicativi molto variabili e non prevedibili, con risorse o con team di sviluppo limitati. Quando si decide di approcciarsi ad un progetto con l'utilizzo di tecnologie serverless, è necessario ponderare bene la scelta del provider di servizi di cloud computing al quale affidarsi. Infatti, lavorare in ambiente serverless può rendere il codice dell'applicazione fortemente dipendente dal provider e si rischia il *lock-in*. Infine, si deve prestare molta attenzione nel caso si debbano sviluppare applicazioni *mission-critical* o che necessitano di una specifica conformità a standard governativi o di settore.

Il provider di servizi di cloud computing scelto per il progetto preso in esame in questa tesi, è *Amazon Web Services* (AWS). Sono stati approfonditi i servizi PaaS e FaaS, messi a disposizione da AWS, che si sono ritenuti utili all'implementazione dell'applicazione.

Per la realizzazione dell'applicativo commissionato, si è iniziato dall'analisi dei requisiti che è stata condotta assieme al cliente ed al business developer. Il progetto in questione si è prestato perfettamente all'utilizzo delle tecnologie serverless data la sua fase di startup, la sua incertezza nel traffico e lo scarso budget allocato. La

prima fase di lavoro, si è basata sulla creazione di mockup del design per ricevere un primo riscontro sia dal punto di vista grafico, che delle funzionalità. Gli stakeholders coinvolti, grazie all'interattività dei mockup creati con Figma, hanno potuto simulare l'utilizzo dell'applicativo ancor prima che venisse realizzato e dare i loro feedback. Per la gestione dell'intero progetto è stata adottata la metodologia Agile, nello specifico il framework Scrum. È stata, quindi, redatta la roadmap dell'intero progetto ed individuate le task che avrebbero composto i 4 Sprint.

Infine, è iniziata la fase di realizzazione del progetto. Durante questa fase sono stati approfonditi ulteriori concetti come *GraphQL* e database *NoSQL* che si sono resi necessari per ottenere il risultato desiderato. Alcune parti previste dal progetto iniziale, come la ricerca con il servizio *Amazon OpenSearch* e la personalizzazione dei contenuti per ogni utente con *Amazon Personalize*, non sono state implementate a causa di un cambio degli obiettivi di business. L'applicativo risultante ha comunque soddisfatto le aspettative del cliente ed i requisiti iniziali del progetto.

Il lavoro svolto durante questa tesi, ha permesso di approfondire le tecnologie serverless ed i servizi forniti dal provider AWS. Inoltre, si è potuto realizzare un progetto reale e cimentarsi nella gestione e coordinazione di un team di sviluppo. Le abilità e le nozioni apprese durante questo periodo, saranno sicuramente sfruttate in progetti futuri.

Appendice A

Mockup Design

Di seguito alcuni Mockup del design dell'applicazione.

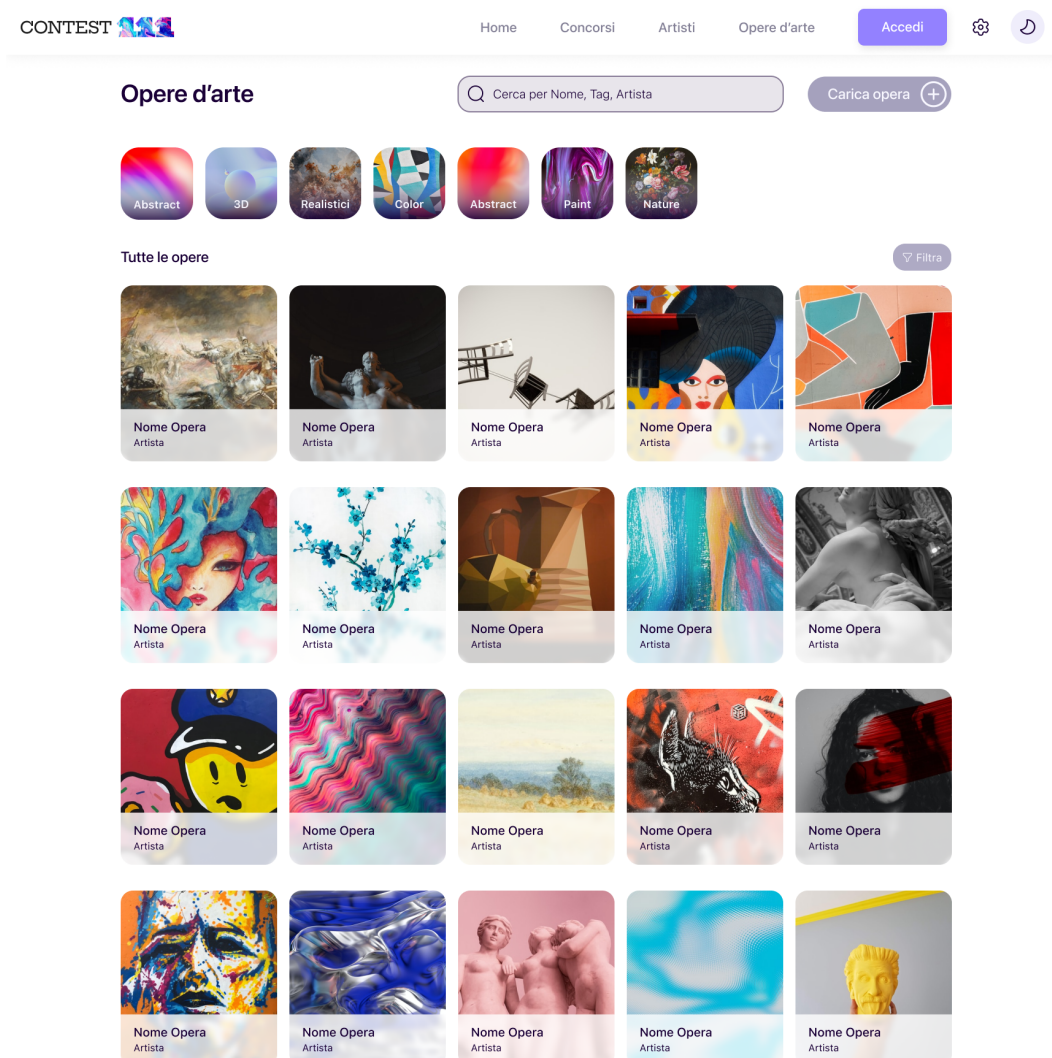






Figura A.1: Mockup Design - Lista opere d'arte

CONTEST  Home Concorsi Artisti Opere d'arte [Accedi](#)  

Opere d'arte > Nome opera ← → Prossima opera



In quale contest desideri votare l'opera?

[Contest 111](#)

[Vota questo progetto](#)

Nome opera

Artista

Descrizione

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Enim a ultrices blandit nisi. Dolor ut odio quis purus, pharetra id. Lectus morbi ipsum in sodales.

Stile
Astratto

Dimensione
50x50 cm

Peso
200g

Contest	Voti	Classifica
Contest 111	2593	156°

Tag

[Arte](#) [Ritratto](#) [Dipinto ad olio](#) [Art](#) [Arte moderna](#) [Colore](#)






[Segnala](#) 

Figura A.2: Mockup Design - Dettagli opera d'arte

CONTEST 


Home Concorsi Artisti Opere d'arte [Accedi](#)  

Opere d'arte > Carica opera



Carica la tua opera





A che concorso desideri partecipare?




Contest 111 

Nome

Descrizione Opera 0/200

Modifica

Stile opera

Dimensione opera

Peso opera

Aggiungi Tag (facoltativo)

Suggeriti

Arte Ritratto Dipinto ad olio Art Arte moderna Colore

Crea tag

[Carica](#)

Figura A.3: Mockup Design - Crea opera d'arte

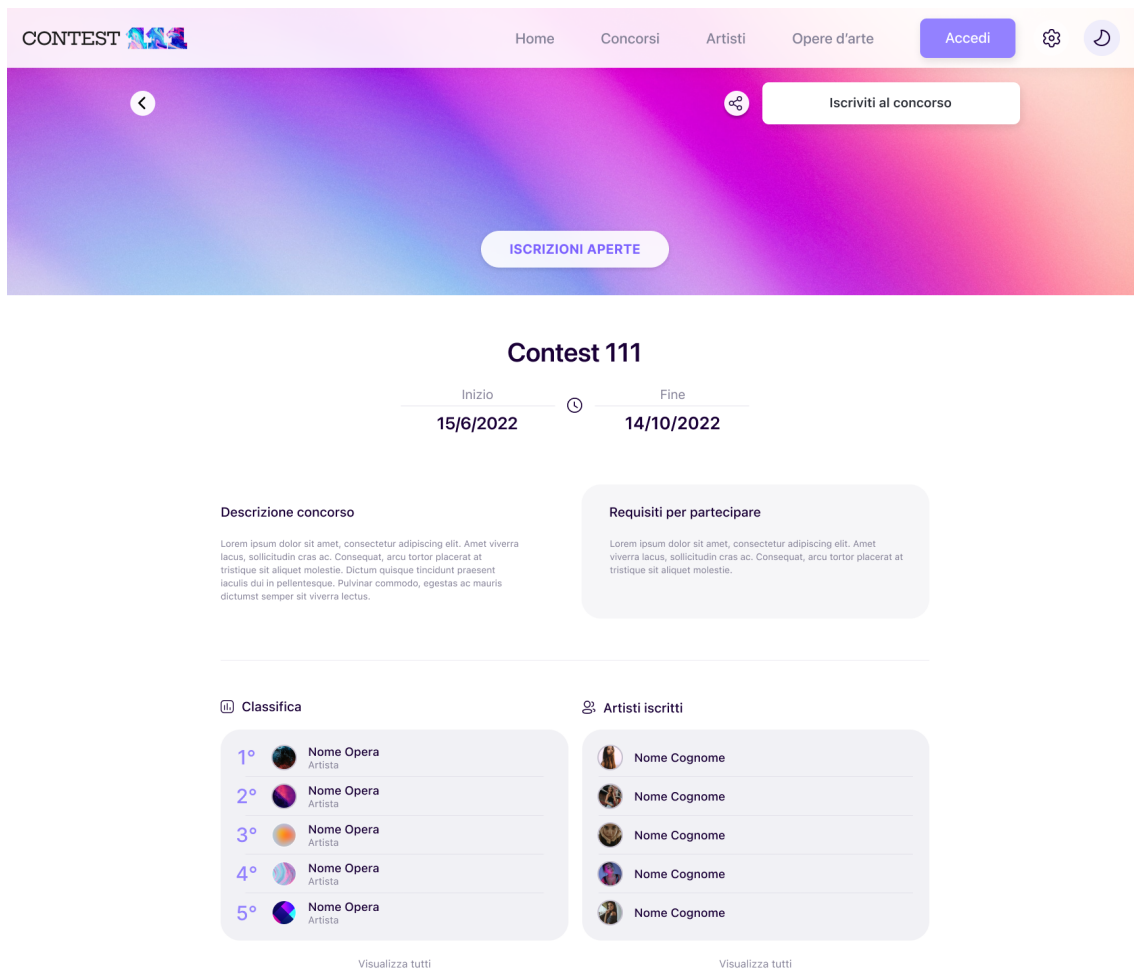


Figura A.4: Mockup Design - Dettagli concorso

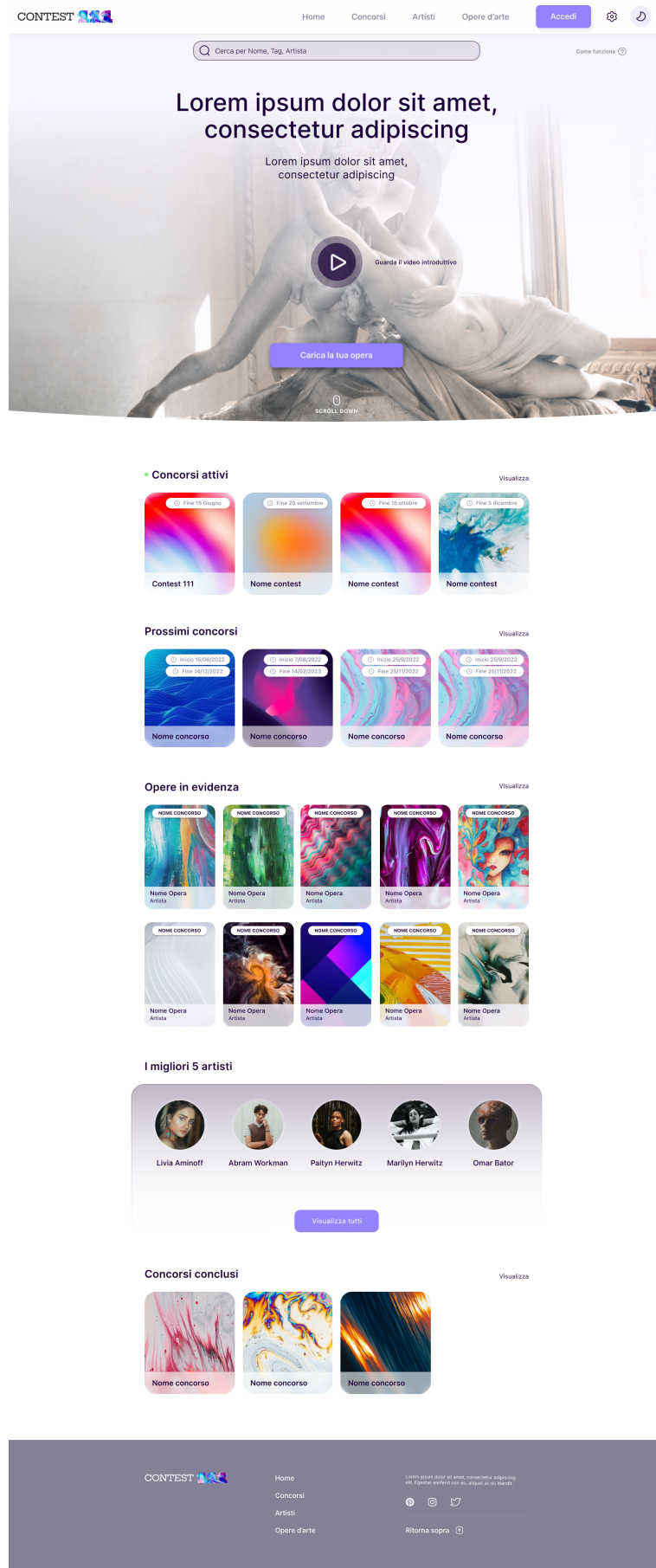


Figura A.5: Mockup Design - Home Page

Appendice B

GraphQL Schema

Di seguito un estratto dello Schema GraphQL dell'applicazione.

```
1 type Artwork {
2   id: ID!
3     @aws_api_key
4     @aws_cognito_user_pools
5   user_id: ID!
6     @aws_api_key
7     @aws_cognito_user_pools
8   title: String!
9     @aws_api_key
10    @aws_cognito_user_pools
11   description: String!
12     @aws_api_key
13     @aws_cognito_user_pools
14   year: Int!
15     @aws_api_key
16     @aws_cognito_user_pools
17   images: [Image!]
18     @aws_api_key
19     @aws_cognito_user_pools
20   video: [Video]
21     @aws_api_key
22     @aws_cognito_user_pools
23   status: ArtworkStatus
24     @aws_api_key
25     @aws_cognito_user_pools
26   artwork_type: ArtworkType
27     @aws_api_key
28     @aws_cognito_user_pools
29   rarity: ArtworkRarity
30     @aws_api_key
31     @aws_cognito_user_pools
32   width: Float
33     @aws_api_key
34     @aws_cognito_user_pools
```

```
35 height: Float
36   @aws_api_key
37   @aws_cognito_user_pools
38 depth: Float
39   @aws_api_key
40   @aws_cognito_user_pools
41 weight: Float
42   @aws_api_key
43   @aws_cognito_user_pools
44 materials: [String]
45   @aws_api_key
46   @aws_cognito_user_pools
47 painting_categories: [String]
48   @aws_api_key
49   @aws_cognito_user_pools
50 painting_styles: [String]
51   @aws_api_key
52   @aws_cognito_user_pools
53 painting_techniques: [String]
54   @aws_api_key
55   @aws_cognito_user_pools
56 photographic_themes: [String]
57   @aws_api_key
58   @aws_cognito_user_pools
59 photographic_currents: [String]
60   @aws_api_key
61   @aws_cognito_user_pools
62 colors: [String]
63   @aws_api_key
64   @aws_cognito_user_pools
65 tags: [String]
66   @aws_api_key
67   @aws_cognito_user_pools
68 timestamp: AWSTimestamp!
69   @aws_api_key
70   @aws_cognito_user_pools
71 }
72
73 enum ArtworkRarity {
74   UNIQUE
75   LIMITED_EDITION
76   OPEN_EDITION
77   UNKNOWN_EDITION
78 }
79
80 enum ArtworkType {
81   PAINTING
82   PHOTOGRAPHY
83   DIGITAL_ART
```



```

84 SCULPTURE
85 }

```

Codice B.1: GraphQL Schema - Artwork

```

1 type ContestSubscription {
2   id: ID!
3   @aws_api_key
4   @aws_cognito_user_pools
5   contest_id: ID!
6   @aws_api_key
7   @aws_cognito_user_pools
8   artwork_id: ID!
9   @aws_api_key
10  @aws_cognito_user_pools
11  user_id: ID!
12  @aws_api_key
13  @aws_cognito_user_pools
14  payment_id: String!
15  @aws_api_key
16  @aws_cognito_user_pools
17  ranking_votes: Int!
18  @aws_api_key
19  @aws_cognito_user_pools
20  timestamp: AWSTimestamp!
21  @aws_api_key
22  @aws_cognito_user_pools
23 }

```

Codice B.2: GraphQL Schema - Subscription

```

1 type Vote {
2   id: ID!
3   contest_id: ID!
4   artwork_id: ID!
5   user_id: ID!
6   value: Int!
7   comment: String
8   payment_id: String!
9   timestamp: AWSTimestamp!
10 }

```

Codice B.3: GraphQL Schema - Vote

```

1 type Mutation {
2   createUser(input: UserCreateFields!): User
3     @aws_cognito_user_pools
4   updateUser(input: UserUpdateFields!): User
5     @aws_cognito_user_pools
6   createArtwork(input: ArtworkCreateFields!): Artwork
7     @aws_cognito_user_pools

```

```

8   updateArtwork(input: ArtworkUpdateFields!): Artwork
9     @aws_cognito_user_pools
10  deleteArtwork(artwork_id: ID!): Artwork
11     @aws_cognito_user_pools
12  subscribeToContest(input: ContestSubscriptionFields!):
13     ContestSubscription
14     @aws_cognito_user_pools
15  subscribeToContestFullData(input:
16     ContestSubscriptionFullDataFields!): ContestSubscription
17     @aws_cognito_user_pools
18  voteArtwork(input: voteArtworkFields!): Vote
19     @aws_cognito_user_pools
20 }

```

Codice B.4: GraphQL Schema - Mutation

```

1  type Query {
2    getUsers(limit: Int, nextToken: String): PaginatedUsers!
3    getUser(user_id: ID!): User
4    getArtists(limit: Int, nextToken: String): PaginatedUsers!
5    getArtist(user_id: ID!): Artist
6    me: User
7     @aws_cognito_user_pools
8    isFreeUsername(username: String!): Boolean!
9    isFreeFiscalCode(fiscal_code: String!): Boolean!
10   getArtworks(filter: ArtworkFilters!, limit: Int, nextToken:
11     String): PaginatedArtworks!
12   getMyArtworks(limit: Int, nextToken: String): PaginatedArtworks!
13     @aws_cognito_user_pools
14   getArtwork(artwork_id: ID!): CompleteArtwork
15   getArtworksByUserId(user_id: ID!, limit: Int, nextToken: String):
16     PaginatedArtworks!
17   getArtworksByContestId(contest_id: ID!, limit: Int, nextToken:
18     String): PaginatedArtworks!
19   getArtistsByContestId(contest_id: ID!, limit: Int, nextToken:
20     String): PaginatedUsers!
21   getContests(filter: ContestFilters!, limit: Int, nextToken:
22     String): PaginatedContests!
23   getContest(contest_id: ID!): Contest
24   getContestsByArtworkId(artwork_id: ID!): [Contest]!
25   getContestRanking(contest_id: ID!, limit: Int, nextToken: String)
26     : PaginatedArtworks!
27   getMySubscriptions(limit: Int, nextToken: String):
28     PaginatedSubscriptions!
29     @aws_cognito_user_pools
30   getMySubscriptionsByContestId(contest_id: ID!, limit: Int,
31     nextToken: String): PaginatedSubscriptions!
32     @aws_cognito_user_pools
33   getSubscriptionsByContestId(contest_id: ID!, limit: Int,
34     nextToken: String): PaginatedSubscriptions!

```

```
26  getSubscriptionsByUserId(user_id: ID!, limit: Int, nextToken:  
    String): PaginatedSubscriptions!  
27  getSubscriptionsByArtworkId(artwork_id: ID!, limit: Int,  
    nextToken: String): PaginatedSubscriptions!  
28  getSubscriptionDetails(subscription_id: ID!): ContestSubscription  
29    @aws_cognito_user_pools  
30 }
```

Codice B.5: GraphQL Schema - Query

Appendice C

Funzioni VTL

Di seguito alcune funzioni VTL dei resolver dell'applicazione.

```
1 {
2   "version" : "2017-02-28",
3   "operation" : "PutItem",
4   "key" : {
5     "id": $util.dynamodb.toDynamoDBJson($util.autoId()),
6     "type": $util.dynamodb.toStringJson("ARTWORK")
7   },
8   "attributeValues" : {
9     "user_id": $util.dynamodb.toStringJson($ctx.identity.sub),
10    "title": $util.dynamodb.toStringJson($ctx.args.input.title),
11    "description": $util.dynamodb.toStringJson($ctx.args.input.description),
12    "year": $util.dynamodb.toNumberJson($ctx.args.input.year),
13    "images": $util.dynamodb.toListJson($ctx.args.input.images),
14    "video": $util.dynamodb.toListJson($ctx.args.input.video),
15    "status": $util.dynamodb.toStringJson($ctx.args.input.status),
16    "artwork_type": $util.dynamodb.toStringJson($ctx.args.input.artwork_type),
17    "rarity": $util.dynamodb.toStringJson($ctx.args.input.rarity),
18    "width": $util.dynamodb.toNumberJson($ctx.args.input.width),
19    "height": $util.dynamodb.toNumberJson($ctx.args.input.height),
20    "depth": $util.dynamodb.toNumberJson($ctx.args.input.depth),
21    "weight": $util.dynamodb.toNumberJson($ctx.args.input.weight),
22    "materials": $util.dynamodb.toListJson($ctx.args.input.materials),
23    "painting_categories": $util.dynamodb.toListJson($ctx.args.input.painting_categories),
24    "painting_styles": $util.dynamodb.toListJson($ctx.args.input.painting_styles),
25    "painting_techniques": $util.dynamodb.toListJson($ctx.args.input.painting_techniques),
26    "photographic_themes": $util.dynamodb.toListJson($ctx.args.input.photographic_themes),
```

```
27     "photographic_currents": $util.dynamodb.toListJson($ctx.args.
input.photographic_currents),
28     "colors": $util.dynamodb.toListJson($ctx.args.input.colors),
29     "tags": $util.dynamodb.toListJson($ctx.args.input.tags),
30     "timestamp": $util.dynamodb.toNumberJson($util.time.
nowEpochMilliseconds())
31 }
32 }
33
34 ## RESPONSE MAPPING
35 #if($ctx.error)
36     $util.error($ctx.error.message, $ctx.error.type)
37 #end
38 $util.toJson($ctx.result)
```

Codice C.1: Funzioni VTL - createArtwork

```
1 {
2     "version" : "2017-02-28",
3     "operation" : "PutItem",
4     "key" : {
5         "id": $util.dynamodb.toDynamoDBJson($util.autoId()),
6         "type": $util.dynamodb.toStringJson("SUBSCRIPTION")
7     },
8     "attributeValues" : {
9         "contest_id" : $util.dynamodb.toStringJson($ctx.args.input.
contest_id),
10        "artwork_id" : $util.dynamodb.toStringJson($ctx.args.input.
artwork_id),
11        "user_id" : $util.dynamodb.toStringJson($ctx.identity.sub),
12        "payment_id" : $util.dynamodb.toStringJson($ctx.args.input.
payment_id),
13        "ranking_votes" : $util.dynamodb.toNumberJson(0),
14        "timestamp": $util.dynamodb.toNumberJson($util.time.
nowEpochMilliseconds())
15    }
16 }
17
18 ## RESPONSE MAPPING
19 #if($ctx.error)
20     $util.error($ctx.error.message, $ctx.error.type)
21 #end
22 $util.toJson($ctx.result)
```

Codice C.2: Funzioni VTL - subscribeToContest

```
1 {
2     "version" : "2017-02-28",
3     "operation" : "PutItem",
4     "key" : {
```

```

5     "id": $util.dynamodb.toDynamoDBJson($util.autoId()),
6     "type": $util.dynamodb.toStringJson("VOTE")
7 },
8 "attributeValues" : {
9     "contest_id" : $util.dynamodb.toStringJson($ctx.args.input.
10    contest_id),
11    "artwork_id" : $util.dynamodb.toStringJson($ctx.args.input.
12    artwork_id),
13    "user_id" : $util.dynamodb.toStringJson($ctx.identity.sub),
14    "payment_id" : $util.dynamodb.toStringJson($ctx.args.input.
15    payment_id),
16    "value" : $util.dynamodb.toNumberJson($ctx.args.input.value),
17    "comment" : $util.dynamodb.toStringJson($ctx.args.input.comment
18    ),
19    "timestamp": $util.dynamodb.toNumberJson($util.time.
20    nowEpochMilliseconds())
21 }
22 }
23 ## RESPONSE MAPPING
24 #if($ctx.error)
25     $util.error($ctx.error.message, $ctx.error.type)
26 #end
27 $util.toJson($ctx.result)

```

Codice C.3: Funzioni VTL - voteArtwork

```

1 {
2     "version" : "2017-02-28",
3     "operation" : "Query",
4     "index" : "artwork-type-index",
5     "query" : {
6         "expression": "artwork_id = :artwork_id AND #type = :type",
7         "expressionNames" : {
8             "#type" : "type"
9         },
10        "expressionValues" : {
11            ":artwork_id" : $util.dynamodb.toDynamoDBJson($ctx.args.input.
12            artwork_id),
13            ":type" : $util.dynamodb.toStringJson("SUBSCRIPTION")
14        }
15    },
16    "filter" : $util.transform.toDynamoDBFilterExpression({
17        "contest_id" : {
18            "eq" : $ctx.args.input.contest_id
19        }
20    })
21 }
22 ## RESPONSE MAPPING

```

```
23 ## Raise a GraphQL field error in case of a datasource invocation
    error
24 #if($ctx.error)
25     $util.error($ctx.error.message, $ctx.error.type)
26 #end
27 ## Launch exception if payment_id already exists.
28 #if($util.toJson($ctx.result.items) != '[]')
29     $util.error("Subscription already exists.", "Subscription Error
        .")
30 #end
31 ## Pass back the result from DynamoDB. **
32 $util.toJson($ctx.result)
```

Codice C.4: Funzioni VTL - checkExistingSubscription

Bibliografia

- [1] *Active Campaign*. URL: <https://www.activecampaign.com/>.
- [2] Akanksha Singh et al. “Overview of PaaS and SaaS and its Application in Cloud Computing”. In: (2016).
- [3] Baldini et al. “Serverless Computing: Current Trends and Open Problems”. In: (2017).
- [4] Mario Villamizar et al. “Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures”. In: (2017).
- [5] Atlassian. *Agile Project Management*. URL: <https://www.atlassian.com/agile/project-management>.
- [6] Atlassian. *Kanban*. URL: <https://www.atlassian.com/agile/kanban>.
- [7] Atlassian. *Scrum*. URL: <https://www.atlassian.com/agile/scrum>.
- [8] Subhankar Dhar. “From outsourcing to Cloud computing: evolution of IT services”. In: ().
- [9] Erwin van Eyk et al. “Serverless is More: From PaaS to Present Cloud Computing”. In: (2018).
- [10] *Figma*. URL: <https://www.figma.com/>.
- [11] CompTIA Inc. *What Is PaaS?* URL: <https://www.comptia.org/content/articles/what-is-paas>.
- [12] Mike Roberts John Chapin. *Programming AWS Lambda*. O’Reilly Media Inc., 2020. ISBN: 9781492041054.
- [13] Federico Masiero. “Cassandra - NoSQL Database”. In: (2019).
- [14] Microsoft. *What is PaaS?* URL: <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-paas/>.
- [15] *Next.js*. URL: <https://nextjs.org/>.
- [16] Claus Pahl. “Containerization and the PaaS Cloud”. In: (2015).
- [17] RedHat. *What is Function-as-a-Service (FaaS)?* 2020. URL: <https://www.redhat.com/en/topics/cloud-native-apps/what-is-faas>.

- [18] 451 Research. “The Carbon Reduction Opportunity of Moving to Amazon Web Services”. In: (2019).
- [19] Gojko Adzic e Robert Chatley. “Serverless Computing: Economic and Architectural Impact”. In: ().
- [20] Mike Roberts. *Serverless Architectures*. 2018. URL: <https://martinfowler.com/articles/serverless.html>.
- [21] *Serverless Land*. URL: <https://serverlessland.com/>.
- [22] Amazon Web Services. *AWS Docs*. URL: <https://docs.aws.amazon.com/>.
- [23] Amazon Web Services. *AWS SDK for JavaScript*. URL: <https://aws.amazon.com/sdk-for-javascript/>.
- [24] Amazon Web Services. *Creating a single-table design with Amazon DynamoDB*. URL: <https://aws.amazon.com/blogs/compute/creating-a-single-table-design-with-amazon-dynamodb/>.
- [25] Amazon Web Services. *Lambda Isolation Technologies*. URL: <https://docs.aws.amazon.com/whitepapers/latest/security-overview-aws-lambda/lambda-isolation-technologies.html>.
- [26] Amazon Web Services. “Optimizing Enterprise Economics with Serverless Architectures”. In: (2022).
- [27] Amazon Web Services. *Serverless and Containers*. URL: <https://docs.aws.amazon.com/whitepapers/latest/logical-separation/serverless-and-containers.html>.
- [28] *Stripe*. URL: <https://stripe.com/it>.
- [29] *TailwindCSS*. URL: <https://tailwindcss.com/>.
- [30] TechBeacon. *The economics of serverless computing: A real-world test*. URL: <https://techbeacon.com/enterprise-it/economics-serverless-computing-real-world-test>.
- [31] TechTarget. *function as a service (FaaS)*. URL: <https://www.techtarget.com/searchitoperations/definition/function-as-a-service-FaaS>.
- [32] TechTarget. *What is PaaS? Platform as a service definition and guide*. URL: <https://www.techtarget.com/searchcloudcomputing/definition/Platform-as-a-Service-PaaS>.
- [33] Vercel. *SWR - React Hooks for Data Fetching*. URL: <https://swr.vercel.app/>.