GRUPPO 2 – SRS

Traccia 1

Andreani Luca	[0001059344]
Ledda Bruno	[0000986619]
Marino Antonio	[0001046877]
Paglia Francesco	[0001034829]

Sommario

1. ABSTRACT

Si vuole realizzare un software in grado di generare servizi web realistici con l'obbiettivo futuro di poter eseguire analisi della sicurezza, analisi dell'affidabilità e stress testing sui servizi generati.

Il software deve quindi essere in grado di:

- Generare siti "*mockup*" di diverso tipo (E-commerce, news, blog, siti aziendali)
- In base al tipo di sito scelto, l'utente deve avere la possibilità di selezionare diversi parametri per la generazione come ad esempio
  - tema, numero progetti (Sito Portfolio)
  - argomenti, quantità e lunghezza dei post (Blog)
  - il numero di prodotti, categorie, ecc. (E-commerce)
  - sezioni, numero di articoli, lunghezza della pagina, ecc. (Sito Web di notizie)
- Generare tutti i file statici
- Automatizzare la distribuzione sul Cloud tramite contenitori
- Gestire in modo sicuro il flusso dei dati

L'obiettivo finale è quindi quello di avere un'API in grado di generare e gestire (avviare, spegnere, modificare) l'ambiente composto dagli "*n*" siti web di diversi tipi creati secondo le caratteristiche definite dall'utente.

2. PIANIFICAZIONE

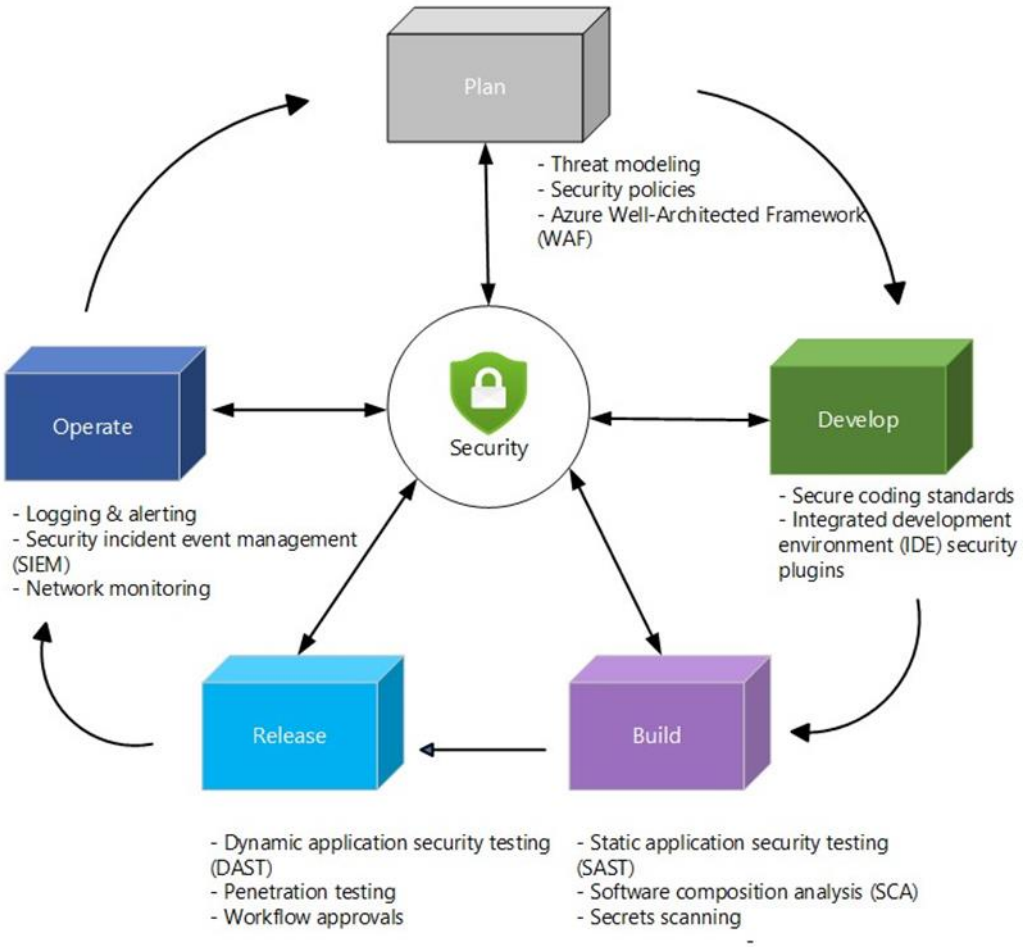
2.1 ADOZIONE DEL CICLO DEV-SEC-OPS

Per la realizzazione del progetto è stato scelto di adottare un approccio DevSecOps in quanto integra al meglio automazione, scalabilità e progettazione del software con particolare attenzione anche alla sicurezza, rendendola una responsabilità condivisa durante tutto il ciclo di vita dell'applicazione.

Seguendo quindi questo principio di garantire una sicurezza completa e costante, è stato deciso di mantenere cicli di sviluppo brevi e frequenti, integrando misure di sicurezza con un'interruzione minima delle operazioni, restando inoltre al passo con le tecnologie innovative come container e micro-servizi, rendendo l'infrastruttura maggiormente scalabile e dinamica.

Inoltre, è stato deciso di adottare un approccio di Security Development Lifecycle (SDL) anche per limitare i costi di gestione e manutenzione del software, in quanto più avanti si va ad agire nella risoluzione di un problema nel ciclo di vita dello sviluppo, più tale correzione costerà.

Questo sarà reso possibile dall'utilizzo di processi e strumenti incorporati in varie pipeline di integrazione continua e distribuzione continua (CI/CD), con particolare attenzione anche alla scalabilità e all'affidabilità del software.



2.2 REQUISITI FUNZIONALI

2.2.1 TABELLA DEI REQUISITI FUNZIONALI

IDENTIFICATIVO	DESCRIZIONE	TIPOLOGIA
R_1F	Ogni Ospite che desidera usufruire completamente dei servizi offerti dall'applicazione deve effettuare una registrazione tramite l'inserimento dei propri dati anagrafici, del proprio indirizzo e-mail e scegliendo un nickname e una password (viene data inoltre la possibilità di registrazione tramite provider quali Google e Microsoft).	Funzionale
R_2F	Ad ogni accesso consecutivo alla registrazione verrà richiesta l'autenticazione della propria identità mediante l'e-mail, la password e un'autenticazione a due fattori.	Funzionale
R_3F	Viene messa a disposizione un'opzione di log-out dal proprio profilo attuale per dare la possibilità di accedere con un altro account esistente.	Funzionale
R_4F	Ogni Utente può accedere alla schermata di creazione di un sito web.	Funzionale
R_5F	In fase di creazione di un qualsiasi sito si devono specificare il template e le relative caratteristiche che si vogliono implementare.	Funzionale
R_6F	Il sistema genera il sito web secondo le specifiche inserite dall'Utente, popolandolo con contenuti generati tramite intelligenza artificiale.	Funzionale
R_7F	Il sistema inserisce il sito web nell'ambiente di gestione dell'Utente che lo ha creato.	Funzionale
R_8F	Ogni Utente deve essere l'unico in grado di accedere al suo ambiente virtuale, dal quale gestire i siti web da lui creati.	Funzionale
R_9F	Il sistema gestisce il deployment del sito web in maniera automatizzata.	Funzionale
$R_{10}F$	Il sistema offre un API che permette di generare e gestire l'ambiente virtuale contenente i siti creati dall'Utente.	Funzionale
$R_{11}F$	Ad ogni Utente può essere associato al massimo un ambiente virtuale.	Funzionale

2.3 REQUISITI NON FUNZIONALI

2.3.1 TABELLA DEI REQUISITI NON FUNZIONALI

IDENTIFICATIVO	DESCRIZIONE	TIPOLOGIA
R_1NF	Il nickname deve essere composto da un minimo di 8 ad un massimo di 20 caratteri alfanumerici, mentre la password da minimo 8 ad un massimo di 20 caratteri alfanumerici.	Non funzionale
R_2NF	La password digitata non deve essere visibile in maniera esplicita sulla schermata.	Non funzionale
R_3NF	All'Utente viene data la possibilità di recupero password tramite e-mail.	Non funzionale
R_4NF	Non deve succedere che un Ospite abbia accesso a servizi riservati a Utente.	Non funzionale
R_5NF	Facilità di inserimento dei parametri di ricerca o necessari alla creazione di un sito.	Non funzionale
R_6NF	Velocità di accesso ai dati in lettura e in scrittura.	Non funzionale
R_7NF	Facilità di navigazione tra le schermate.	Non funzionale
R_8NF	Buona formattazione grafica dei dati.	Non funzionale
R_9NF	Gestione e protezione del flusso dei dati	Non funzionale
R_{10}NF	Sistemi, dati e applicazioni replicabili	Non funzionale
R_{11}NF	Sistemi e applicazioni scalabili elasticamente dipendentemente dal carico di richieste	Non funzionale
R_{12}NF	Garantire elevata disponibilità	Non funzionale
R_{13}NF	Realizzazione di un sistema affidabile	Non funzionale
R_{14}NF	Realizzazione di un sistema resiliente	Non funzionale
R_{15}NF	Realizzazione di un sistema sicuro	Non funzionale
R_{16}NF	Sistema pensato per gli utenti del mercato europeo	Non funzionale
R_{17}NF	Rispettare il budget a disposizione, prevedendo una spesa massima di 250€ al mese	Non funzionale

2.3.2 REPLICAZIONE

La replicazione è basata sulla creazione di copie di server, reti, dati e applicazioni in maniera ridondante. Tramite la replicazione è possibile ottenere anche altre proprietà come la scalabilità, la disponibilità, l'affidabilità e le performance.

Nel nostro progetto, effettueremo una replicazione geografica dei server per aumentare la disponibilità e diminuire la latenza, assicurando agli utenti una connessione veloce al IDC più vicino alla loro posizione. Naturalmente, questo tipo di replicazione sarà supportata dall'integrazione di *dispatcher* geografici e locali.

I dati saranno replicati impiegando una consistenza debole, dato che la nostra applicazione non gestisce dati che necessitano di una consistenza forte. Nello specifico, adotteremo un approccio moderno basato sui database non relazionali (NoSQL) che garantiscono la consistenza transazionale. Quest'ultima garantisce scalabilità elevata e distribuzione su larga scala.

L'ultima tipologia di replicazione che vogliamo perseguire è quella software. Innanzitutto, svilupperemo un'applicazione basata su un'architettura a microservizi. In seguito, useremo un meccanismo di orchestrazione automatico che sarà in grado di replicare i microservizi e scalarli dinamicamente sulla base del carico di lavoro del sistema. Maggiore sarà la richiesta di un determinato servizio e maggiori saranno i nodi che lo forniranno.

2.3.3 SCALABILITA'

La scalabilità elastica (o elasticità) è la capacità del sistema di gestire carichi di lavoro che possono aumentare o diminuire in modo variabile nel corso del tempo, con un uso efficace delle risorse. Affinché un'architettura sia scalabile, bisogna evitare la presenza di colli di bottiglie e di singoli punti di fallimento tramite la replicazione e l'isolamento dei fallimenti.

Per far ciò, nella fase di progettazione abbiamo usato come riferimento il cubo della scalabilità:

- Asse X per la replicazione dei servizi e dei dati: replicare risorse hardware e software in maniera tale da avere più istanze su cui distribuire il carico di richieste degli utenti. Ovviamente, questo tipo di replicazione incrementa anche la disponibilità. Uno dei possibili limiti potrebbe essere dato dalla replicazione della persistenza, risolto però dalla scelta di adottare un database NoSQL.
- Asse Y per la decomposizione funzionale: creare applicazioni basate su microservizi. Così facendo potremmo scalare ogni servizio sulla base delle richieste e dei nodi disponibili.
- Asse Z per la divisione dei servizi e dei dati: ogni server esegue lo stesso codice ma su un sottoinsieme di dati diversi. Questo diventa possibile grazie alla creazione di *pod* geografici e allo smistamento delle richieste degli utenti verso il *pod* più vicino alla loro posizione. Grazie a ciò, possiamo ridurre la latenza della rete e ottenere tempi di risposta più veloci.

Testeremo la scalabilità dei nostri sistemi tramite una simulazione di carichi crescenti. Quest'ultima considera il numero di utenti concorrenti, il numero di richieste inviate e il volume dei dati da gestire.

2.3.4 DISPONIBILITA'

La disponibilità rappresenta la percentuale di tempo in cui un sistema, un'infrastruttura o un servizio restano operativi e usufruibili da parte degli utenti. Questa proprietà è importante soprattutto quando si parla di sistemi *mission critical*, ovvero l'insieme dei sistemi il cui funzionamento è essenziale per la sopravvivenza del business o dell'organizzazione. Per quanto riguarda il nostro progetto, individueremo quali sono le nostre risorse mission critical e, successivamente, ne analizzeremo la disponibilità.

2.3.5 AFFIDABILITA'

L'affidabilità è la capacità di un sistema o di un componente di funzionare in certe condizioni per un determinato periodo di tempo. Disponibilità e affidabilità possono risultare molto simili fra di loro, ma non lo sono: la prima è una metrica relativa al sistema, mentre la seconda è orientata verso il cliente perché fornisce informazioni sulla sua percezione della qualità del servizio.

Nel corso del nostro progetto, ci focalizzeremo sulle seguenti tre fasi per riuscire a realizzare un sistema che sia altamente affidabile:

1. Prevenzione dei problemi tramite una buona progettazione, implementazione e test.
2. Monitoraggio continuo del funzionamento delle risorse e delle applicazioni.
3. Manutenzione e capacità di rispondere agli incidenti.

All'interno della fase di progettazione useremo il cubo della disponibilità:

- Asse X per la scalabilità: replicare le risorse per incrementarne la disponibilità. La replicazione non solo ci assicura la scalabilità, ma ci garantisce anche la ridondanza. Questa ci consente di avere un insieme di componenti uguali e paralleli che offrono lo stesso servizio e, se uno di questi dovesse smettere di funzionare, ci sarebbero sempre gli altri attivi.
- Asse Y per la dipendenza tra i servizi: esprime la dipendenza tra i servizi che sono legati fra di loro da una catena di chiamate sincrone. In questo caso, se un servizio non funzionasse correttamente, influenzerebbe tutti gli altri servizi a lui legato: il guasto non sarebbe isolato. Bisogna evitare le chiamate sincrone tra i servizi perché diminuiscono la disponibilità.

- Asse Z per l'isolamento dei guasti: simile all'asse Z del cubo della scalabilità dove si vanno a creare dei sottoinsiemi geografici o dei segmenti di clientela per gestire separatamente eventuali problemi che potrebbero verificarsi, isolandoli dall'intera rete.

Il nostro lavoro si concentrerà in particolar modo sulla realizzazione di un sistema che riesca a individuare, isolare e risolvere nel minor tempo possibile varie tipologie di problemi, dando la sensazione all'utente che tutto funzioni sempre correttamente.

2.3.6 RESILIENZA

La resilienza descrive la capacità di un sistema per auto-guarirsi, riprendersi e continuare ad operare dopo aver fronteggiato guasti e problemi di vario genere (bug, cyberattacchi, catastrofi naturali, ecc.). Nella resilienza sono compresi la continuità del business, le tecniche di *disaster recovery* e la gestione dei guasti, con l'obiettivo di ridurre la portata e la durata degli eventi distruttivi.

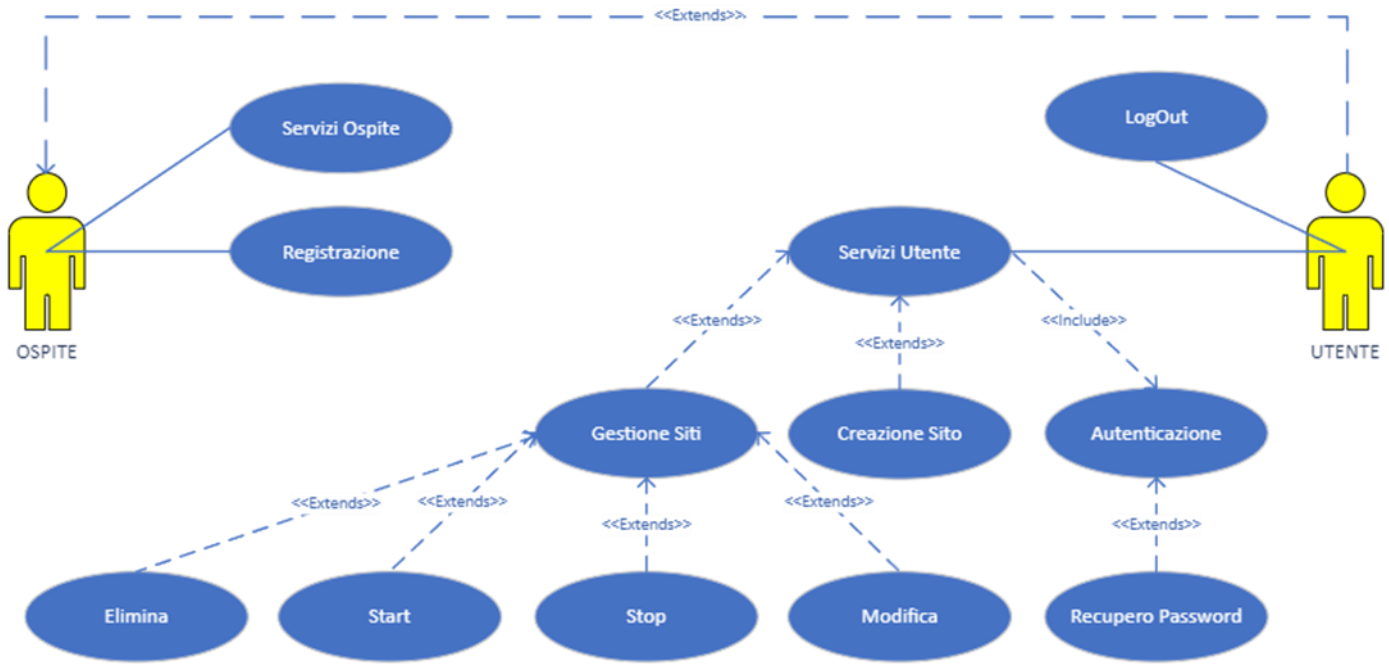
Ci concentreremo in maniera particolare sulle 4R:

- Riconoscimento: il sistema e i suoi operatori devono riconoscere gli indicatori che precedono il fallimento del sistema.
- Resistenza: attuare delle strategie di resistenza in grado di resistere ai problemi tecnici e agli attacchi informatici, riducendo la probabilità che il sistema fallisca.
- Ripristino: in caso di fallimento, bisogna prevedere delle attività di ripristino che facciano tornare operativi i sistemi critici nel minor tempo possibile, riducendo i disservizi per gli utenti.
- Reintegrazione: i servizi tornano a funzionare normalmente.

L'idea di base è che non possiamo evitare tutti i fallimenti del sistema e i cyberattacchi. Di conseguenza, dobbiamo limitare gli effetti di questi eventi e puntare a tornare operativi.

2.4 CASI D'USO

2.4.1 DIAGRAMMA DEI CASI D'USO



2.4.2 SCENARI DEI CASI D'USO

Titolo	SERVIZI OSPITE
Descrizione	Servizio che permette all'Ospite di inserire i propri dati e registrarsi nell'applicazione
Attori	Ospite, Utente
Relazioni	
Precondizioni	
Post-condizioni	
Scenario principale	1. L'Ospite/Utente apre l'applicazione 2. L'Ospite/Utente ha accesso al servizio di Registrazione 3. L'Ospite/Utente ha accesso al servizio di Login
Scenari alternativi	
Requisiti non funzionali	Integrità e sicurezza dei dati, facilità di navigazione delle schermate.

Titolo	REGISTRAZIONE
Descrizione	Servizi offerti dall'applicazione per tutti gli Utenti
Attori	Ospite
Relazioni	
Precondizioni	L'Ospite non è registrato sull'applicazione
Post-condizioni	L'Ospite è registrato sull'applicazione
Scenario principale	1. L'applicazione mostra all'Ospite una schermata in cui inserire dati per la registrazione 2. L'Ospite inserisce i propri dati 3. Il sistema presenta la schermata di accesso
Scenari alternativi	Scenario A: *È già presente un Utente con le stesse credenziali* 1. Il sistema notifica l'Ospite 2. Il sistema ricarica la schermata di registrazione Scenario B: *Dati inseriti non sono validi* 1. Viene segnalato che i dati inseriti non sono validi 2. Il sistema ricarica la schermata di registrazione
Requisiti non funzionali	Integrità e sicurezza dei dati, facilità di navigazione delle schermate. Ogni Utente deve essere identificato in maniera univoca tramite il nickname. Velocità di memorizzazione dei dati.

Titolo	LOG-OUT
Descrizione	Il sistema permette all'Utente di disconnettersi dall'applicazione
Attori	Utente
Relazioni	
Precondizioni	L'Utente deve aver effettuato l'autenticazione
Post-condizioni	L'Utente non è più autenticato
Scenario principale	1. L'Utente tramite il pulsante corrispondente si disconnette dall'applicazione 2. Il sistema mostra la schermata di benvenuto
Scenari alternativi	
Requisiti non funzionali	Integrità e sicurezza dei dati, facilità di navigazione delle schermate.

Titolo	AUTENTICAZIONE
Descrizione	Modalità di accesso all'applicazione da parte dell'Utente
Attori	Utente
Relazioni	Servizi Utente, Recupero Password
Precondizioni	
Post-condizioni	L'Ospite è registrato e autenticato
Scenario principale	1. Presentazione schermata di login 2. L'Utente inserisce le proprie credenziali 3. Il sistema verifica le credenziali e queste risultano corrette 4. Il sistema invia le credenziali per l'autenticazione a due fattori 5. L'Utente inserisce le credenziali per l'autenticazione a due fattori 6. Il sistema verifica le credenziali e queste risultano corrette 4. Viene mostrata la schermata principale (Home Utente)
Scenari alternativi	Scenario A: *Il nickname non è presente nel sistema* 1. Viene mostrato un avviso 2. Il sistema mostra la schermata di accesso al sistema Scenario B: *Password non valida* 1. Viene mostrato un avviso 2. Il sistema mostra la schermata di accesso al sistema Scenario C: *Credenziali autenticazione a due fattori non valide* 1. Viene mostrato un avviso 2. Il sistema mostra la schermata di accesso al sistema Scenario D: *Password dimenticata* 1. Il sistema mostra la scherma di reset della password
Requisiti non funzionali	Integrità e sicurezza dei dati: la password non deve essere visibile sulla schermata. Velocità ricerca dati.

Titolo	RECUPERO PASSWORD
Descrizione	Servizio mediante il quale un Utente può resettare la password di accesso al suo profilo
Attori	Utente
Relazioni	Autenticazione
Precondizioni	
Post-condizioni	L'Utente ha modificato la password di accesso
Scenario principale	1. Il sistema mostra la schermata di recupero password 2. L'Utente inserisce il proprio indirizzo e-mail 3. Il sistema controlla l'e-mail e risulta associata ad un profilo Utente 4. Il sistema invia una e-mail all'Utente contenente il servizio di modifica password 5. Il sistema mostra la schermata di log-in
Scenari alternativi	1. Il sistema mostra la schermata di recupero password 2. L'Utente inserisce il proprio indirizzo e-mail 3. Il sistema controlla l'e-mail e non risulta associato ad un profilo Utente 4. Il sistema notifica l'errore 5. Il sistema ricarica la schermata di recupero password
Requisiti non funzionali	Integrità e sicurezza dei dati. Velocità ricerca dati.

Titolo	SERVIZI UTENTE
Descrizione	Schermata offerta dall'applicazione riservati agli Utenti
Attori	Utente
Relazioni	Autenticazione, Creazione Sito, Gestione Sito
Precondizioni	
Post-condizioni	L'Ospite è in grado di utilizzare tutte le funzioni disponibili
Scenario principale	1. Autenticazione effettuata con successo 2. L'Utente accede alla schermata principale (Home Utente) 3. Il sistema permette l'accesso ai servizi a tutti i servizi offerti dall'applicazione
Scenari alternativi	
Requisiti non funzionali	Facilità di navigazione delle schermate. Integrità e sicurezza dei dati.

Titolo	CREAZIONE SITO
Descrizione	Servizio mediante il quale un Utente può creare un nuovo sito web
Attori	Utente
Relazioni	Servizi Utente
Precondizioni	
Post-condizioni	Creazione e deployment automatizzato di un nuovo sito web.
Senario principale	1. Il sistema mostra la schermata di inserimento dei dati necessari alla creazione di un nuovo sito web. 2. L'Utente inserisce i dati necessari alla creazione del sito. 3. Il sistema verifica i dati e questi sono corretti. 4. Il sistema genera un nuovo sito popolandolo con contenuti generati da un'intelligenza artificiale. 5. Il sistema si occupa del deployment del sito. 6. Il sistema aggiunte il sito all'ambiente gestionale dell'Utente. 7. Il sistema mostra la schermata principale (Home Utente).
Scenario alternativo	Scenario A: *I dati necessari alla creazione del sito inseriti dall'Utente non rispettano le specifiche di sistema.* 1. Il sistema notifica l'errore. 2. Il sistema ricarica la schermata di creazione del sito. Scenario B: *La creazione del sito non va a buon fine.* 1. Il sistema notifica l'errore. 2. Il sistema ricarica la schermata di creazione del sito. Scenario C: *Il deployment del sito non va a buon fine.* 1. Il sistema notifica l'errore. 2. Il sistema ricarica la schermata di creazione del sito.
Requisiti non funzionali	Facilità di navigazione delle schermate. Integrità e sicurezza dei dati.

Titolo	GESTIONE SITI
Descrizione	Servizio mediante il quale un Utente può visualizzare e gestire i siti da lui creati
Attori	Utente
Relazioni	Servizi Utente, Start, Stop, Modifica, Elimina
Precondizioni	
Post-condizioni	L'Utente accede alla schermata "Gestione Siti"
Scenario principale	1. L'Utente accede alla schermata "Gestione Siti" 2. Il sistema mostra la lista dei siti e permette l'accesso ai servizi: • Start • Stop • Modifica • Elimina
Scenari alternativi	
Requisiti non funzionali	Facilità di navigazione delle schermate. Integrità e sicurezza dei dati.

Titolo	ELIMINA
Descrizione	Servizio mediante il quale un Utente può eliminare un sito web del quale è il proprietario
Attori	Utente
Relazioni	Gestione Siti
Precondizioni	L'Utente deve essere il proprietario del sito web
Post-condizioni	Il sito web viene eliminato
Scenario principale	1. L'Utente conferma la volontà di eliminare il sito web 2. Il sistema elimina il sito web 3. Il sistema notifica l'eliminazione all'Utente 4. Il sistema mostra la schermata principale (Home Utente)
Scenari alternativi	Scenario A: *l'eliminazione non è andata a buon fine* 1. Il sistema notifica l'errore 2. Il sistema mostra la schermata principale (Home Utente) Scenario B: *l'Utente non acconsente all'eliminazione* 1. L'Utente nega la volontà di eliminare il sito 2. Il sistema mostra la schermata principale (Home Utente)
Requisiti non funzionali	Facilità di navigazione delle schermate. Integrità e sicurezza dei dati.

Titolo	START
Descrizione	Servizio mediante il quale un Utente avvia l'ambiente di hosting del sito web
Attori	Utente
Relazioni	Gestione Siti
Precondizioni	L'Utente deve essere il proprietario del sito web
Post-condizioni	L'ambiente di hosting del sito viene avviato
Scenario principale	1. L'Utente conferma la volontà di avviare l'ambiente di hosting 2. Il sistema avvia l'ambiente di hosting 3. Il sistema notifica l'avvio dell'ambiente all'Utente 4. Il sistema mostra la schermata principale (Home Utente)

Scenari alternativi	Scenario A: *l'avvio dell'ambiente non è andata a buon fine* 1. Il sistema notifica l'errore 2. Il sistema mostra la schermata principale (Home Utente)
Requisiti non funzionali	Facilità di navigazione delle schermate. Integrità e sicurezza dei dati.

Titolo	STOP
Descrizione	Servizio mediante il quale un Utente arresta l'ambiente di hosting del sito web
Attori	Utente
Relazioni	Gestione Siti
Precondizioni	L'Utente deve essere il proprietario del sito web
Post-condizioni	L'ambiente di hosting del sito viene arrestato
Scenario principale	1. L'Utente conferma la volontà di arrestare l'ambiente di hosting 2. Il sistema arresta l'ambiente di hosting 3. Il sistema notifica l'arresto dell'ambiente all'Utente 4. Il sistema mostra la schermata principale (Home Utente)
Scenari alternativi	Scenario A: *l'arresto dell'ambiente non è andata a buon fine* 5. Il sistema notifica l'errore 6. Il sistema mostra la schermata principale (Home Utente)
Requisiti non funzionali	Facilità di navigazione delle schermate. Integrità e sicurezza dei dati.

Titolo	MODIFICA
Descrizione	Servizio mediante il quale un Utente può modificare un sito web
Attori	Utente
Relazioni	Gestione Siti
Precondizioni	L'Utente deve essere il proprietario del sito web
Post-condizioni	Il sito web viene modificato
Scenario principale	1. Il sistema avvia l'ambiente di modifica del sito web. 2. L'utente richiede le modifiche da apportare al sito web. 3. Il sistema si occupa di apportare le modifiche al sito web. 4. Il sistema si occupa del deployment del sito web modificato 5. Il sistema notifica l'avvenuta modifica del sito all'Utente 6. Il sistema mostra la schermata principale (Home Utente)
Scenari alternativi	Scenario A: *la modifica del sito non è andata a buon fine* 1. Il sistema notifica l'errore 2. Il sistema mostra la schermata principale (Home Utente)
Requisiti non funzionali	Facilità di navigazione delle schermate. Integrità e sicurezza dei dati.

3. REALIZZAZIONE

3.1 TEAM DEV

3.1.1 MICROSERVIZI

L'applicazione è stata sviluppata in modo da prevedere una suddivisione in servizi più piccoli e indipendenti che comunicano tra loro in modo flessibile e scalabile tramite il protocollo HTTP e il pattern architetturale REST. Grazie alle API il frontend e il backend dell'applicazione comunicano tra loro consentendo una separazione delle responsabilità, obbiettivo raggiunto grazie anche all'utilizzo dell'approccio MVC all'interno del framework che permette dunque lo sviluppo del codice con una maggiore modularità. Tra i servizi implementati vi sono:

- Servizio autenticazione (gestito da un controller che esegue operazioni CRUD sul database)
- Servizio "User environment" che permette:
    - Creazione/eliminazione sito web sul cloud
    - Generazione contenuti delle pagine tramite servizi esterni
    - Caricamento file statici sul cloud
    - Abilitazione/disabilitazione sito web

Le API REST sono basate su protocolli standard e possono essere facilmente consumate da diverse tecnologie e piattaforme. Inoltre, consentono una separazione chiara tra il servizio che espone l'API e il servizio che la consuma, promuovendo così l'indipendenza e la modularità dei microservizi.

3.1.2 SCELTA DEL FRAMEWORK (Node.js)

Node.js è una tecnologia che supporta la creazione e l'esecuzione di app e servizi Web scalabili e ad alte prestazioni. La scelta è ricaduta su questo framework perché:

- Fornisce un ambiente di programmazione coerente e orientato agli oggetti, indipendentemente dal fatto che il codice oggetto venga archiviato ed eseguito in locale, eseguito in locale ma distribuito sul Web o eseguito in remoto
- Fornisce un ambiente di esecuzione del codice che:
    - Riduce al minimo i conflitti di distribuzione e controllo delle versioni del software
    - Promuove l'esecuzione sicura del codice, incluso il codice creato da terze parti sconosciute o semi-attendibili
    - Elimina i problemi di prestazioni degli ambienti con script
    - È particolarmente efficiente in termini di scalabilità, grazie alla programmazione asincrona e non bloccante
- Crea tutte le comunicazioni sugli standard di settore per garantire che il codice si integri con qualsiasi altro codice
- Fornisce servizi di base quali gestione della memoria, gestione di thread e servizi remoti, imponendo una rigida indipendenza dai tipi e altre forme di accuratezza del codice che garantiscono sicurezza ed efficienza

3.1.3 INTEGRAZIONE INTELLIGENZA ARITIFICIALE

Per quanto riguarda la fase di popolazione dei template, dopo un'attenta analisi, la scelta più adatta per il servizio da usare, e ricaduta sull'API di OpenAI. Il servizio, dopo la generazione e la configurazione di una chiave associata ad un account specifico per il progetto, consente di definire tramite una chiamata asincrona:

- Il modello da utilizzare per la generazione dei testi, nel nostro caso specifico "text-davinci-003"
- Il prompt relativo al contenuto specifico che si vuole generare

- Un numero massimo di token da generare (usato anche come parametro) per la generazione di contenuti di diversa lunghezza, sulla base delle specifiche dichiarate dall'utente
- Un parametro chiamato "temperature" che ha il compito di determinare la "*randomicitá*" del contenuto generato

Il modello che è stato scelto è particolarmente performante nel "*text completion*" e permette una generazione di massimo 4097 token e massimo 60 richieste al minuto. Inoltre, ha un costo di $0.02 ogni 1000 token (token generati + token presenti nel prompt), che lo rendono bilanciato per quanto riguarda il rapporto prestazioni/costi.

Alcuni parametri generati da questo servizio vengono poi passati come prompt ad un secondo servizio, chiamato Google Search Console API, che restituisce un URL relativo all'immagine che soddisfa maggiormente la richiesta. Questa API viene utilizzata per visualizzare, aggiungere o rimuovere proprietà e Sitemap, eseguire query avanzate per i dati dei risultati della ricerca Google secondo le specifiche di un motore di ricerca personalizzato nel quale e possibile definire quali siti includere e quali escludere per la ricerca.

3.1.4 TEST

All'interno della pipeline di CI/CD, abbiamo modificato il workflow in modo tale che l'applicazione e le Azure Functions, venissero distribuite sia sull'ambiente di sviluppo, sia sull'ambiente di test.

Ad esempio, nel job "test-env", abbiamo incluso l'esecuzione di due script in Phyton per testare se le chiamate alle funzioni gestiscono correttamente gli errori relativi a paramenti sbagliati nella richiesta, ed un altro per monitorare la corretta esecuzione di ciascuna funzione.

Mentre per quanto riguarda la pipeline dell'applicazione, dopo il deploy in ambiente di test, vengono eseguiti alcuni script in Javascript per testare le funzionalità di base come:

- La corretta connessione al Database
- La corretta disponibilità dei pacchetti aggiuntivi di node.js
- Test di compatibilità del browser
- Il corretto inserimento dei caratteri in fase di autenticazione
- La disponibilità dei segreti per la generazione dei token di autenticazione

È importante eseguire i test non solo nell'ambiente di test ma anche in quello di produzione per testare le nuove modifiche in un ambiente sottoposto a un carico realistico.

3.2 TEAM OPS

3.2.1 INFRASTRUTTURA

Definire correttamente l'infrastruttura è uno dei pilastri su cui si fonda la realizzazione di un prodotto di qualità. Su Azure tutti gli aspetti infrastrutturali sono gestiti tramite Azure Resource Manager (ARM). Abbiamo utilizzato questo servizio per scegliere e configurare le seguenti risorse:

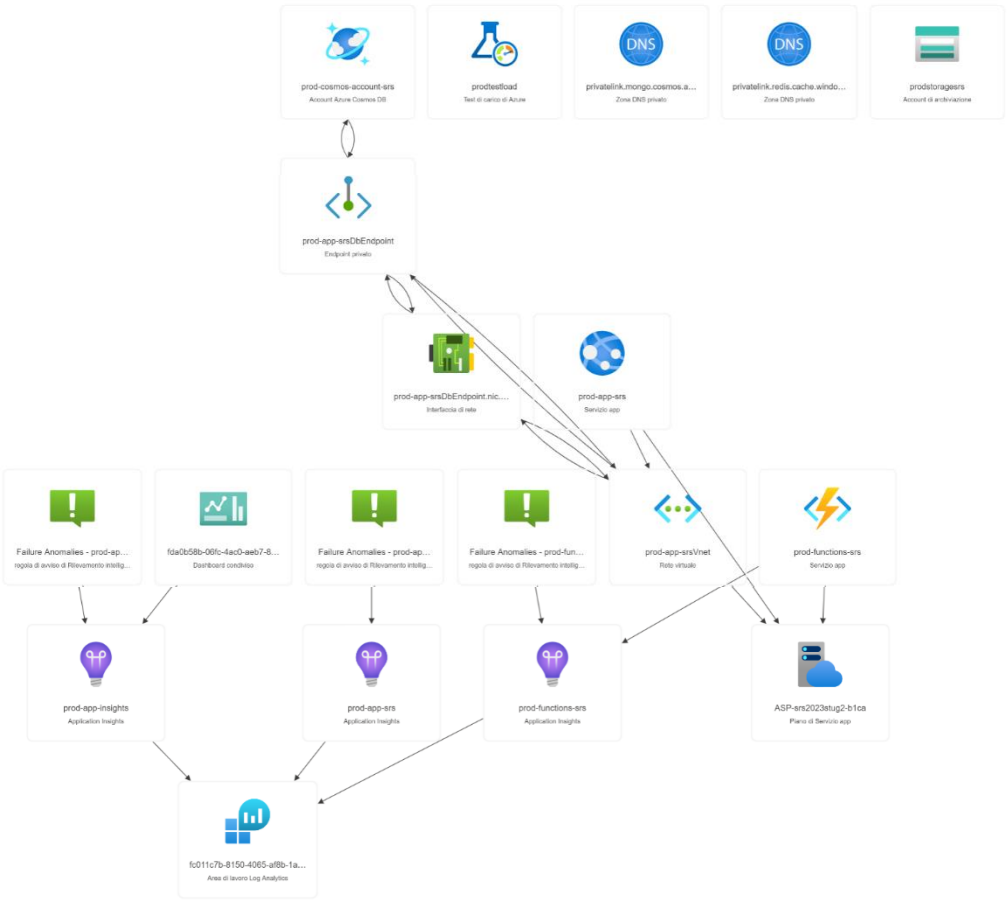
- Compute services: risorse su cui esegue un'applicazione.
    - App Web: ottimizzata per gestire al meglio applicazioni e servizi web. Questa risorsa ospita l'applicazione web, scritta interamente in Node.Js e usufruibile dagli utenti tramite apposite interfacce web.
    - Functions: pensata per l'esecuzione di funzioni *event-driven*. L'infrastruttura cloud fornisce tutte le risorse necessarie per mantenere in esecuzione l'applicazione, sollevandoci dall'onere di gestirne la configurazione. Questa risorsa ospita le funzioni Python, all'interno delle quali ci saranno tutti i comandi utili per gestire direttamente le risorse degli utenti. Queste funzioni eseguiranno occasionalmente, solamente quando verranno triggerate.
- Storage services: risorse usate per memorizzare dati semi-strutturati e non strutturati.
  Tra i vari servizi offerti dalla risorsa account di storage, usiamo specialmente il *Blob* (*Binary Large Object*). Quest'ultimo offre la possibilità di ospitare un sito web statico all'interno di un container dedicato chiamato $web. Il sito è raggiungibile tramite l'utilizzo di un *url* univoco e sarà possibile modificare i relativi file semplicemente caricandoli nell'apposito container, sfrutteremo proprio questa funzionalità per ospitare i siti web generati dagli utenti.
  L'altra funzionalità dell'account di storage che sfruttiamo è la CDN, in maniera tale da migliorare notevolmente l'esperienza utente tramite una fruizione più rapida dei contenuti richiesti.
- Database services: risorse usate per memorizzare dati strutturati (SQL) e semi-strutturati (NoSQL). Tra le varie opzioni disponibili, abbiamo scelto Cosmos DB perché offre un database NoSQL con tutti i vantaggi che ne conseguono in termini di replicazione, scalabilità e disponibilità. Questi ultimi garantiscono tempi di accesso molto ridotti e maggiore sicurezza in caso di incidenti.
- Networking service: risorse usate per gestire la comunicazione nelle reti interne e da/verso il mondo esterno. Tra i componenti utilizzati troviamo:
    - Rete virtuale: usata per costruire una rete interna sicura che consenta la comunicazione tra le varie risorse di uno specifico gruppo di risorse.
    - Application gateway: usato per effettuare il dispatching locale e ridistribuire il carico di richieste HTTP localmente su tutti i nodi in cui esegue l'applicazione web. Possiamo implementare questo tipo di dispatching semplicemente tramite l'impiego di uno switch.
    - Front door: usato per effettuare il dispatching geografico e ridirezionare le richieste HTTP globali verso il back-end più vicino alla posizione del richiedente.
    - Content Delivery Network (CDN): progettata per l'invio rapido e affidabile di audio, video, immagini e altri file agli utenti tramite i server più vicini a loro, aumentando in modo significativo la velocità e la disponibilità. La implementeremo nei siti web dei nostri utenti in maniera tale da fornire loro un servizio di maggiore qualità.

La scelta delle risorse e la relativa configurazione è stata fatta a seguito della fase di pianificazione e nel pieno rispetto di tutte le esigenze di ogni team coinvolto. In particolare, abbiamo usato la documentazione di Azure per valutare le caratteristiche di ogni risorsa.

Tutte le risorse selezionate sono contenute in tre gruppi di risorse. Ogni gruppo di risorse rappresenta un ambiente e ha le sue peculiarità:

- Sviluppo: ambiente usato soprattutto dal team Dev per lo sviluppo del codice.
- Test: ambiente usato per testare automaticamente o manualmente il software appena realizzato. I test possono essere eseguiti sia su singole unità (unit test) sia su interi processi operativi.
- Produzione: ambiente che contiene l'applicazione che gli utenti utilizzano.

Negli ambienti di sviluppo e test abbiamo usato il più possibile delle risorse basiche e gratuite; mentre nell'ambiente di produzione abbiamo impiegato delle risorse premium che ci garantiscono scalabilità automatica, resilienza, affidabilità, disponibilità e sicurezza.



Inoltre, abbiamo creato un quarto e ultimo gruppo di risorse che usiamo come contenitore logico per tutti i siti web statici dei nostri utenti, rilegandoli in un ambiente dedicato e separato dagli ambienti con cui interagiamo direttamente. Un approccio multi-ambiente ci consente di creare, testare e rilasciare il codice con maggiore velocità e frequenza, rendendo la distribuzione il più semplice possibile.

3.2.2 INFRASTRUCTURE AS A CODE

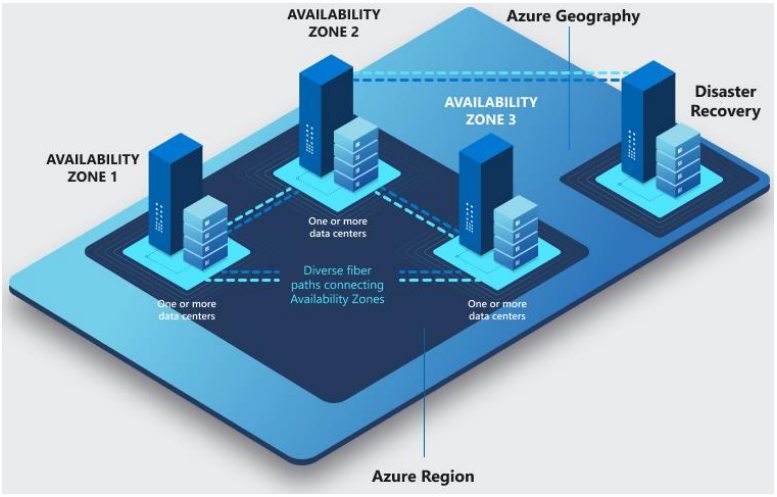
La gestione manuale degli ambienti e delle risorse al loro interno è sicuramente un processo arduo e incline a errori. Per far fronte a ciò, nel nostro progetto abbiamo adottato l'Infrastructure as Code (IaC) per gestire la complessità infrastrutturale e la distribuzione dei vari ambienti.

IaC ci consente di gestire tutte le risorse tramite dei file dichiarativi, evitando la configurazione manuale che potrebbe portare diversi problemi se un domani decidessimo di ridistribuire lo stesso ambiente automaticamente. Inoltre, l'adozione del IaC garantisce l'idempotenza, ovvero la capacità di produrre sempre lo stesso risultato a partire dall'esecuzione della stessa operazione. Questa caratteristica consente di ottenere sempre lo stesso ambiente a seguito di una sua distribuzione.

Per il nostro progetto abbiamo deciso di implementare la IaC servendoci di file JSON, scritti seguendo le notazioni direttamente interpretabili da ARM. Inoltre, ogni qual volta avverrà una modifica al template di un ambiente, partirà una pipeline dedicata che controlla automaticamente se ci sono errori all'interno dei file JSON ed effettua la distribuzione dell'ambiente modificato direttamente su Azure. All'interno di questa pipeline possiamo persino includere degli strumenti di sicurezza. Grazie a tutto ciò, riusciamo a garantire la consistenza e miglioriamo la produttività del team.

3.2.3 REPLICAZIONE

La prima tipologia di replicazione (replication) su cui abbiamo lavorato è quella dei server che, nel nostro progetto, sono rappresentati dalla risorsa App Web. Essa supporta la ridondanza di zona che ci consente di avere sempre almeno tre istanze dell'applicazione web in tre differenti IDC presenti nella stessa regione. Come regione principale abbiamo scelto l'Europa occidentale, ovvero i Paesi Bassi. Questa regione è accoppiata con la regione dell'Europa settentrionale (Irlanda), all'interno della quale è presente una copia della nostra App Web, pronta ad essere usata in caso di disastri nella regione principale. Grazie alla ridondanza di zona e all'accoppiamento, forniamo ai nostri utenti più punti di accesso alla nostra applicazione web e garantiamo loro la continuità del nostro servizio anche in caso di gravi incidenti.



La replicazione dei dati è garantita tramite l'utilizzo di database non relazionali (NoSQL) che ci consentono di ottenere una scalabilità elevata e una distribuzione su larga scala. Il costo da pagare per ottenere questi vantaggi è la consistenza transazionale, ovvero la garanzia che la consistenza venga ottenuta solo a un certo istante futuro indefinito. Da qui consegue che, i sistemi NoSQL potrebbero fornire anche dati inconsistenti che dovranno essere gestiti opportunatamente lato software. All'interno del nostro progetto, per gestire i dati e la relativa replicazione, usiamo Cosmos DB. Questo database è distribuito a livello globale, il che consente di avere tante repliche locali del database in cui andare a leggere e scrivere i dati. La replicazione avviene in maniera trasparente, garantendo bassa latenza, scalabilità elastica, disponibilità elevata e coerenza dei dati.

La ridondanza dei dati si applica anche agli account di storage, di cui si salverà la copia primaria nella regione principale e la copia secondaria nella regione accoppiata.

Parlando di replicazione, è importante sottolineare che agli utenti vorremmo offrire la possibilità di attivare la funzionalità di Content Delivery Network per i loro siti web statici. A seguito dell'attivazione, i siti web degli utenti saranno connessi alla CDN di Azure e sarà possibile salvare i relativi contenuti statici all'interno degli Edge server di Microsoft sparsi per tutto il globo. Questo velocizza l'accesso ai siti web da parte degli utenti e riduce il carico di richieste ai server principali.

3.2.4 SCALABILITA' ELASTICA

La scalabilità elastica (*elasticity*) è fondamentale per garantire ottime prestazioni e alta disponibilità. La scalabilità comprende sia l'istanziazione di nuove risorse per rispondere ai picchi di carico, sia la rimozione di risorse superflue per adattarsi ai cali di richieste.

Azure supporta la scalabilità verticale e orizzontale, ma predilige la seconda perché si adatta meglio ai cambi di carico, può essere eseguita automaticamente, costa meno e garantisce maggiore affidabilità. Inoltre, Azure offre una serie di servizi che sono in grado di scalare automaticamente.

Questo tipo di ridimensionamento è basato sulla raccolta delle metriche delle risorse (utilizzo della CPU, della memoria, ecc.) e delle applicazioni (richieste in coda, richieste soddisfatte in un secondo, ecc.), che servono per analizzare meglio il carico a cui il sistema è sottoposto.

Tra le risorse che adoperiamo nel progetto, quelle che supportano la scalabilità automatica sono:

- App Web e Functions: abbiamo connesso queste due risorse allo stesso piano di fatturazione. Per questo piano abbiamo definito delle regole di scalabilità orizzontale automatica che monitorano l'utilizzo della CPU e, tutte le volte che questo valore sale al di sopra del 70%, Azure aggiunge un nuovo server e crea nuove istanze dell'applicazione. La scalabilità orizzontale automatica va da un minimo di 3 nodi fino a un massimo di 30 e dipende completamente dal carico a cui sono sottoposti i server.
- Cosmos DB: elevata scalabilità garantita dalla consistenza transazionale e dalla possibilità di distribuire le istanza del database globalmente. È possibile scalare automaticamente anche il throughput in base al carico.

È importante comprendere che, per quanto le operazioni di ridimensionamento possano essere veloci, sarà sempre necessario del tempo per portarle a compimento. Di conseguenza, in questo intervallo tecnico dove il sistema è sottoposto a un sovraccarico, potrebbero verificarsi dei peggioramenti nelle performance.

3.2.5 DISPONIBILITA'

La disponibilità (availability) rappresenta un'altra delle proprietà su cui ci siamo soffermati nella realizzazione del nostro progetto. Innanzitutto, è importante specificare che le nostre risorse *mission critical* sono le App Web, le Functions, il Cosmos DB e gli account di storage. Se queste risorse non dovessero funzionare a dovere, rischieremmo un blocco nella fornitura del nostro servizio e, in alcuni casi, la perdita permanente dei dati. Questi incidenti potrebbero ledere persino la nostra reputazione.

Una volta individuate le risorse *mission critical*, abbiamo analizzato i *Service Level Agreement* e la documentazione di Microsoft Azure per scoprire la disponibilità dei servizi che usiamo:

- App Web: disponibilità garantita al 99,95%.
- Functions: disponibilità garantita al 99,95%.
- Cosmos DB: disponibilità quasi continua (99,995% per scritture e 99,999% per letture) grazie alla replicazione dei database di scrittura in due regioni e di quelli di lettura globalmente. Questo ci garantisce che, nel caso in cui dovessero esserci problemi in una regione, il traffico sarebbe automaticamente e in maniera trasparente ridirezionato verso le regioni in cui ci sono le copie funzionanti del database.
- Account storage: disponibilità del 100% (undici 9) che potrebbe calare solo in caso di incidenti temporanei. Una disponibilità così elevata è data dal fatto che molti servizi dipendono dallo storage. Infatti, quest'ultimo viene considerato una risorsa critica che deve essere disponibile e funzionante praticamente in qualsiasi momento.

Possiamo sinterizzare le disponibilità soprastanti sfruttando la scala dei '9'.

Risorsa	Disponibilità [%]	Downtime annuale	Downtime mensile	Downtime giornaliero
App web - Functions	99,95	4,38 ore	21,92 minuti	43,20 secondi
Cosmos DB (s)	99,995	26,30 minuti	2,19 minuti	4,32 secondi
Cosmos DB (l)	99,999	5,26 minuti	26,30 secondi	0,86 secondi
Storage account	Undici '9'	315,36 microsecondi	25,92 microsecondi	864 nanosecondi

Per aumentare la disponibilità, potremmo far ricorso alla replicazione e a una corretta gestione di quest'ultima tramite appositi algoritmi di dispatching.

3.2.6 AFFIDABILITA'

L'affidabilità (reliability) è tra le proprietà più importanti che un sistema deve avere. Innanzitutto, per realizzare un sistema affidabile siamo partiti dalla sua progettazione, garantendo sempre la ridondanza.

Dal punto di vista software, usiamo un'architettura basata su microservizi e le chiamate a questi sono fatte in maniera asincrona. Questa tipologia di architettura ci permette di scalare singolarmente i vari servizi e di isolare i malfunzionamenti, affinché il malfunzionamento di un servizio non impatti sul funzionamento degli altri. Ciò è reso possibile anche grazie all'impiego di chiamate asincrone che evitano la creazione di forti dipendenze software.

Dal punto di vista infrastrutturale, una grande mano ci viene data dalla piattaforma Azure che gestisce la replicazione delle varie risorse automaticamente, asincronicamente e dipendentemente dal carico del sistema. Inoltre, per la parte di archiviazione fornisce svariate soluzioni basate sulla consistenza debole che consente di replicare e distribuire i dati più facilmente.

Sostanzialmente, sfruttiamo il più possibile la ridondanza perché ci assicura persino l'isolamento dei guasti, ovvero la garanzia che un incidente venga confinato in un servizio o in una risorsa specifica, senza propagarsi all'intero sistema causando un disservizio generale.

Nel nostro caso, l'isolamento coinvolge tutte le risorse, ad eccezione di quelle che gestiscono il traffico di rete come load balancer e firewall. Il corretto isolamento dei guasti è ottenibile solo se il sistema è stato implementato per essere affidabile e se avviene un continuo monitoraggio di quest'ultimo. Infatti, solo grazie al monitoraggio costante diventa possibile individuare i guasti, attivare tempestivamente le misure necessarie per risolverli e ritornare operativi nel minor tempo possibile.

Le risorse che abbiamo usato (App Web, Functions, Cosmos DB e account storage) sono considerate affidabili. Innanzitutto, tutte sfruttano la ridondanza.

Ad esempio, l'applicazione Web ha sempre un minimo di 3 repliche attive in 3 zone diverse della regione principale e 1 replica standby in un'altra regione secondaria. Stesso discorso vale per Cosmos DB che ha 2 repliche attive del database di scrittura in due regioni separate e molte repliche attive del database di lettura distribuite in tutto il globo.

Inoltre, per ognuna di queste risorse sono disponibili vari strumenti per monitorarle e valutarne lo stato di salute. In caso di guasti, Azure provvede autonomamente ad attivare tutte le misure necessarie per ritornare operativi il prima possibile.

I tempi di ripristino sono relativamente bassi: Cosmos DB ha un Recovery Time Objective (RTO – tempo necessario per far tornare un'applicazione operativa a seguito di un guasto) di meno di 15 minuti.

Infine, le risorse che usiamo sono ampiamente e automaticamente scalabili, caratteristica che ci torna molto utile quando bisogna replicare parte del software o dell'hardware per incrementare la disponibilità a seguito di un guasto.

Da tutto ciò è possibile evincere che, anche se dovessero esserci gravi problemi a una o più di queste risorse, il sistema non dovrebbe bloccarsi completamente e riusciremmo a garantire la continuità del nostro servizio agli utenti, dando loro l'impressione che tutto funzioni sempre e correttamente.

3.2.7 **RESILIENZA**

Anche se abbiamo progettato e implementato tutti gli accorgimenti atti a rendere il nostro sistema affidabile, scalabile e sicuro, gli incidenti potrebbero sempre capitare. Investire sulla resilienza ci consente di individuare, valutare e gestire gli incidenti in maniera sicura ed efficace, evitando gravi conseguenze.

Nell'ambiente operativo potremmo dover far fronte a problemi come:

- Latenza di rete superiore al normale;
- Errori di connessione;
- Crash di un processo che dovrà essere riavviato o spostato;
- Sovraccarico di un servizio che non consente la risposta in tempi brevi;
- Guasti hardware;
- Cyberattacchi.

Per essere pronti ad affrontare a tutto ciò, abbiamo previsto un piano di disaster recovery. Come prima cosa, abbiamo previsto dei backup per tutti i servizi che usufruiscono della persistenza come Cosmos DB e gli account di archiviazione. I backup sono eseguiti periodicamente e vengono salvati nella regione accoppiata (Europa settentrionale) alla nostra regione principale (Europa occidentale). Tutto il codice relativo alla App Web e alle Functions è conservato su GitHub e possiamo facilmente ridistribuirlo. Stesso discorso vale per tutta la parte infrastrutturale visto che viene gestita tramite l'Infrastructure as Code.

Inoltre, abbiamo realizzato il sistema con le seguenti caratteristiche:

- Ridondante: più istanze per lo stesso servizio per evitare il single point of failure; gestione delle richieste tramite load balancer per bilanciare i carichi in base alle disponibilità; geo-replicazione per essere sicuri di avere sempre almeno una copia funzionante anche in caso di calamità in una regione.
- Scalabile: utilizzo dei microservizi per scalare l'applicazione; utilizzare componenti che siano in grado di scalare orizzontalmente e proporzionalmente al carico; usare la scalabilità automatica.
- Affidabile: isolamento dei guasti; monitoraggio continuo.
- Sicuro: integrazione della sicurezza in qualsiasi fase del ciclo DevOps.

Le caratteristiche appena elencate sono di fondamentale importanza perché conferiscono al sistema la capacità di adattarsi e di rispondere alle situazioni più critiche. Sicuramente, la piattaforma di Azure ci ha fornito un grande aiuto sotto questo punto di vista, dato che la gestione dei problemi software e il ripristino delle risorse hardware avviene in maniera praticamente trasparente e automatica.

3.3 TEAM SEC

3.3.1 ANALISI DEI RISCHI – THREAT MODELING

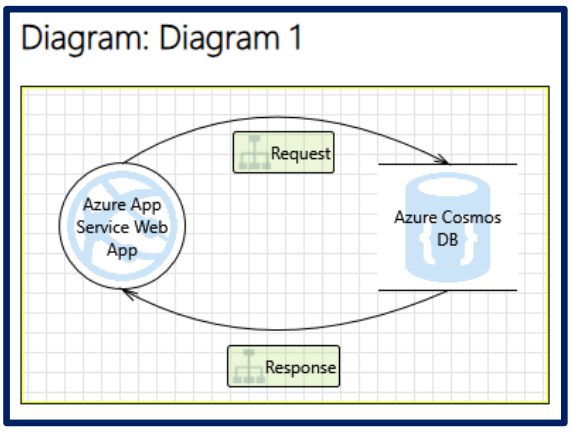
Nella fase di pianificazione è fondamentale stabilire un'analisi dei rischi, in quanto consente di attenuare i problemi di sicurezza e privacy del progetto.

Definite le risorse, i dati e il flusso di informazioni, abbiamo eseguito una modellazione delle minacce utilizzando Microsoft Threat Modeling Tool, uno strumento che consente di identificare e ridurre tempestivamente potenziali problemi di sicurezza, quando sono relativamente semplici e convenienti da risolvere.

Inoltre, per la modellazione delle minacce, vengono prese in considerazione i primi 10 rischi per la sicurezza descritti in OWASP (Open Web Application Security Project).

Di conseguenza, i costi totali di sviluppo vengono notevolmente ridotti.

Di seguito viene riportato un breve estratto del report fornito da Microsoft Threat Modeling Tool e una tabella nella quale vengono esaminate le possibili minacce da mitigare:



1. A compromised access key may permit an adversary to have more access than intended to an Azure Cosmos DB instance [State: Not Started] [Priority: High]

Category:	Elevation of Privileges
Description:	A compromised access key may permit an adversary to have over-privileged access to an Azure Cosmos DB instance
Justification:	<no mitigation provided>
Possible Mitigation(s):	Use resource (SAS like) tokens (derived using master keys) to connect to Cosmos DB instances whenever possible. Scope the resource tokens to permit only the privileges necessary (e.g. read-only). Store secrets in a secret storage solution (e.g. Azure Key Vault). Refer: https://aka.ms/tmt-th54
SDL Phase:	Design

2. An adversary can gain unauthorized access to Azure Cosmos DB instances due to weak network security configuration [State: Not Started] [Priority: High]

Category:	Elevation of Privileges
Description:	An adversary can gain unauthorized access to Azure Cosmos DB instances due to weak network security configuration
Justification:	<no mitigation provided>
Possible Mitigation(s):	Restrict access to Azure Cosmos DB instances by configuring account-level firewall rules to only permit connections from selected IP addresses where possible. Refer: https://aka.ms/tmt-th57
SDL Phase:	Implementation

3. An adversary having access to Azure Cosmos DB may read sensitive clear-text data [State: Not Started] [Priority: High]

Category:	Information Disclosure
Description:	An adversary having access to Azure Cosmos DB may read sensitive clear-text data
Justification:	<no mitigation provided>
Possible Mitigation(s):	Encrypt sensitive data before storing it in Azure Document DB.
SDL Phase:	Design

4. An adversary may reuse a stolen long-lived resource token, access key or connection string to access an Azure Cosmos DB instance [State: Not Started] [Priority: High]

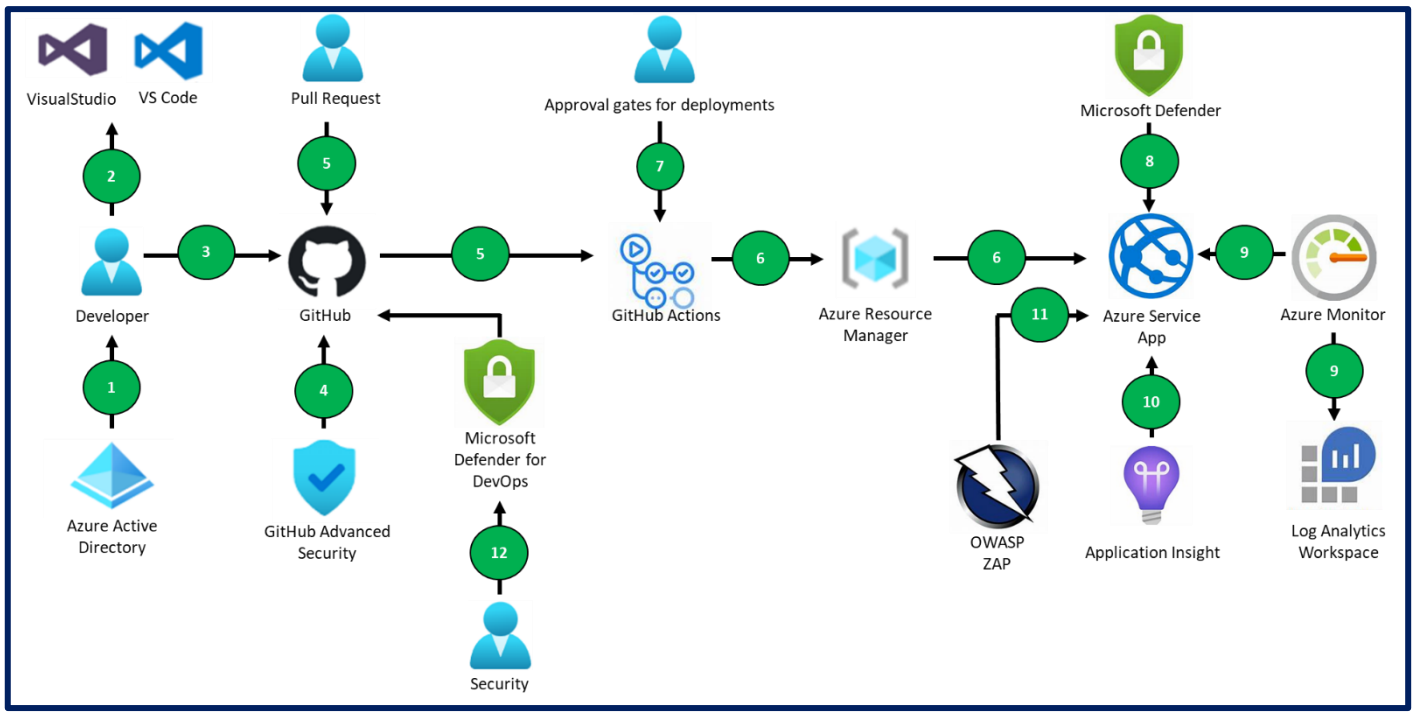
Category:	Elevation of Privileges
Description:	An adversary may reuse a stolen long-lived resource token, access key or connection string to access an Azure Cosmos DB instance
Justification:	<no mitigation provided>
Possible Mitigation(s):	Use minimum token lifetimes for generated resource tokens. Rotate secrets (e.g. resource tokens, access keys and passwords in connection strings) frequently, in accordance with your organization's policies. Refer: https://aka.ms/tmt-th55
SDL Phase:	Implementation

Dall'analisi del report fornito da Microsoft Threat Modeling Tool, e dall'analisi delle risorse, delle iterazioni degli utenti e del flusso dei dati sono quindi emerse queste possibili minacce da mitigare per rendere il più sicuro possibile il nostro sistema:

Minaccia	Proprietà di sicurezza	Mitigazione
Spoofing	Autenticazione	Richiedere autenticazioni HTTPS
Manomissione	Integrità	Convalidare i certificati SSL/TLS
		Usare i certificati di Azure
Ripudio	Non ripudio	Abilitazione del monitoraggio e diagnostica di Azure.
Diffusione di informazioni	Riservatezza	Crittografare i dati sensibili inattivi e in transito.
Denial of Service	Disponibilità	Monitorare le metriche delle prestazioni
		Implementare i filtri di connessione
		Abilitare la protezione DDos di Azure
Elevazione dei privilegi	Autorizzazione	POLP tramite l'utilizzo di Azure Active directory
Phishing	Autenticazione	Implementare un'autenticazione a due fattori
		Gestione degli accessi tramite Azure Active Directory

Successivamente all'analisi dei rischi, in fase di pianificazione, è stato scelto di mitigare le potenziali minacce individuate seguendo un protocollo di progettazione, sviluppo e distribuzione preciso, integrando la sicurezza ad ogni passo del ciclo di sviluppo del progetto.

3.3.2 FLUSSO DEL PROCESSO – INTEGRAZIONE DELLA SICUREZZA NEL CICLO DEV-OPS



1. Azure Active Directory (Azure AD) è configurato come provider di identità per GitHub.
2. Gli sviluppatori usano Visual Studio Code o Visual Studio con estensioni di sicurezza abilitate per analizzare in modo proattivo il codice per le vulnerabilità di sicurezza.
3. Gli sviluppatori eseguono il commit del codice dell'applicazione in un repository GitHub di proprietà dell'azienda.
4. GitHub integra l'analisi automatica della sicurezza e delle dipendenze tramite GitHub Advanced Security.
5. Le richieste pull attivano compilazioni di integrazione continua (CI) e test automatizzati tramite GitHub Actions.
6. Il flusso di lavoro di compilazione CI tramite GitHub Actions genera un'immagine del contenitore Docker archiviata in Azure Service App.
7. È possibile introdurre approvazioni manuali per le distribuzioni in ambienti specifici, ad esempio la produzione, come parte del flusso di lavoro di recapito continuo (CD) in GitHub Actions.
8. Microsoft Defender viene usato per analizzare Azure Service App, Key Vault di Azure e per le vulnerabilità di sicurezza.
    a. Analizza l'immagine del contenitore per le vulnerabilità di sicurezza note al caricamento in Azure Service App.
9. Azure Monitor:
    a. Informazioni dettagliate sui contenitori recuperando le metriche delle prestazioni.
    b. I log di diagnostica e dell'applicazione vengono estratti in un'area di lavoro di Azure Log Analytics per eseguire query di log.
10. Application Insights: è un servizio di gestione delle prestazioni che consente di monitorare un'applicazione Web live.
11. Open Web Application Security Project (OWASP ZAP) è uno strumento open-source utilizzato per eseguire test di penetrazione.
12. Defender per DevOps è un servizio che consente al team di sicurezza di gestire la sicurezza DevOps in ambienti multi-pipeline (GitHub).

3.3.3 POLP (Principle Of Least Privilege)

Abbiamo deciso di adottare criteri di identità come perimetro di sicurezza primario, estendendo questo principio anche ai team che lavorano al progetto (responsabili dello sviluppo, responsabili della sicurezza, responsabili del deployment), oltre che agli utenti che utilizzeranno il servizio.

Il controllo degli accessi e dei privilegi avviene in maniera centralizzata tramite Azure Active Directory, assegnando sempre i minimi privilegi necessari differenziati in base alla tipologia di utente o collaboratore del progetto.

3.3.4 AUTENTICAZIONE E ACCESSO JUST-IN-TIME

L'autenticazione viene gestita quindi in maniera centralizzata (Azure Active Directory), in quanto l'utilizzo di meccanismi di autenticazione e autorizzazione forniti dalla piattaforma Azure invece di un codice personalizzato risulta più robusto e meno soggetto ad errori di programmazione.

Per rafforzare notevolmente il processo è stato deciso di usufruire di un'autenticazione a due fattori, implementata seguendo inoltre il principio dell'accesso Just-In-Time, riducendo ulteriormente il tempo di esposizione dei privilegi, assegnando ai token di autorizzazione una scadenza temporale.

3.3.5 AGGIORNAMENTO DEI COMPONENTI UTILIZZATI

Per cercare di evitare di inserire nuove vulnerabilità introdotte negli strumenti utilizzati, è necessario eseguire continuamente l'inventario dei componenti lato client e lato server (ad esempio framework e librerie) e le relative dipendenze per gli aggiornamenti.

Per monitorare le dipendenze dei componenti utilizzati, e la versione corretta in uso, abbiamo utilizzato GitHub Dependabot, uno strumento di scansione automatizzata.

3.3.6 RIDUZIONE DELLA SUPERFICIE D'ATTACCO

La superficie di attacco è la somma totale della posizione in cui possono verificarsi potenziali vulnerabilità. Cercando quindi di ridurre al minimo la superficie d'attacco, in fase di revisione abbiamo rimosso:

- Codice per le funzionalità non ancora rilasciate
- Interfacce di rete e protocolli non usati o deprecati
- Risorse inutilizzate

Inoltre, i dati di produzione, in quanto non devono essere usati per lo sviluppo o il test dell'applicazione vengono necessariamente separati in un ambiente dedicato, riducendo la superficie d'attacco.

3.3.7 GESTIONE DELLE CHIAVI – SEGRETI – CERTIFICATI

Abbiamo scelto di usufruire del servizio di Azure per la gestione, dei segreti e dei certificati.

Azure risolve in maniera centralizzata:

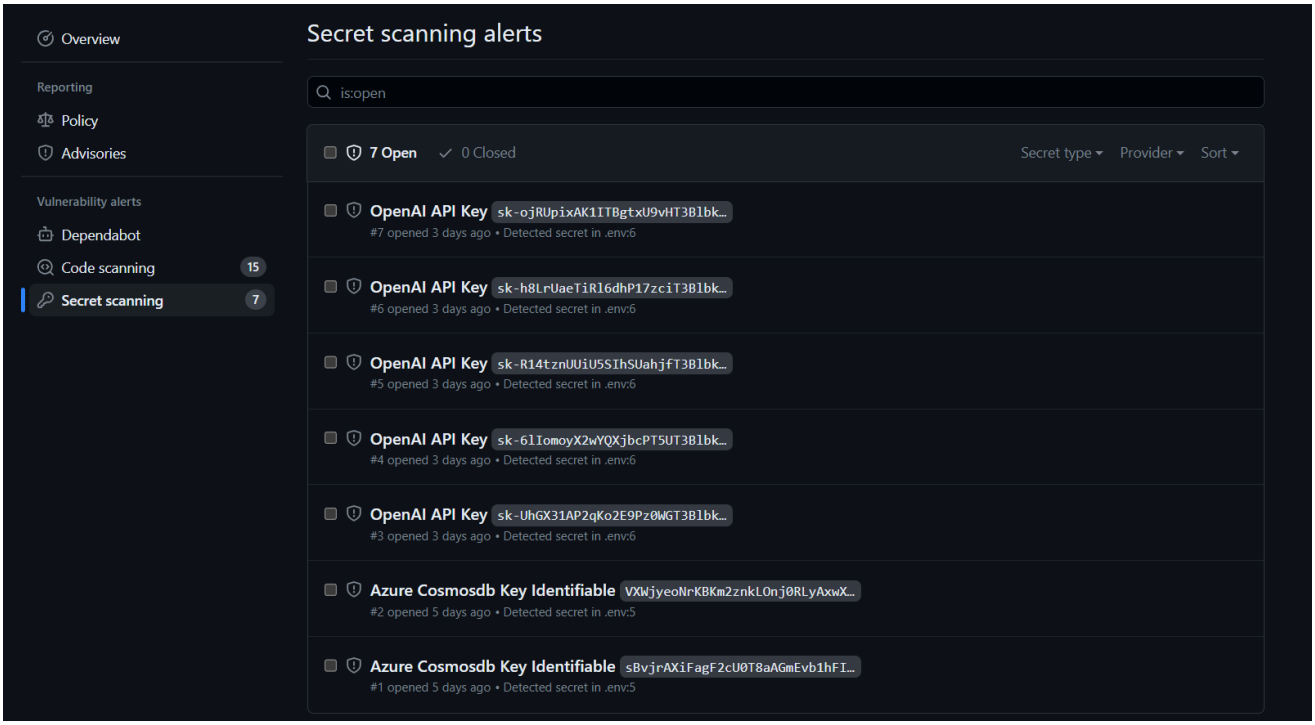
- Gestione dei segreti: archivia in modo sicuro e controlla rigorosamente l'accesso a token, password, certificati e chiavi API
- Gestione delle chiavi: offre una soluzione centralizzata di gestione delle chiavi, permettendo di creare e controllare le chiavi di crittografia usate per crittografare i dati
- Gestione dei certificati: facilita la gestione e la distribuzione di certificati TLS/SSL pubblici e privati da usare con Azure e fra le risorse interne

I segreti vengono archiviati in singoli insiemi di credenziali. Ogni insieme di credenziali ha i propri criteri di configurazione e sicurezza per controllare l'accesso. È possibile accedere ai dati tramite un'API REST.

3.3.8 ANALISI DEI SEGRETI

L'analisi dei segreti avviene mediante due strumenti principali:

- GitHub Advanced Security: esegue periodicamente anche un'analisi completa del contenuto esistente nei repository e invia notifiche di avviso.
- Defender for Cloud: rileva credenziali, segreti, certificati e altri contenuti sensibili nel codice sorgente e nell'output della compilazione.



3.3.9 PROTEZIONE DEI DATI SENSIBILI

La protezione dei dati è una parte essenziale della strategia di sicurezza. In seguito alla classificazione (categorizzazione) dei dati archiviati in base alla riservatezza e all'impatto aziendale, abbiamo seguito alcune strategie di protezione come:

- Evitare segreti in "*hardcode*"
- Crittografare i dati sensibili
- Eseguire controlli di accesso e autenticazione ai dati
- Utilizzare protocolli SSL/TLS per il flusso dei dati in transito

3.3.10 CONVALIDA E VERIFICA DEGLI INPUT DELL'APPLICAZIONE

Per proteggere l'applicazione da eventuali attacchi di injection, consideriamo tutti gli input come non attendibili, in quanto parametri nell'URL, input dell'utente, dati provenienti dal database o da un'API potrebbe essere stato potenzialmente manipolato.

Il sistema deve quindi convalidare che i dati siano sintatticamente e semanticamente validi prima che l'applicazione li usi in qualunque modo, inclusa la visualizzazione all'utente.

La convalida degli input avviene all'inizio del flusso di dati, per garantire che solo i dati in formato corretto entrino nel flusso di lavoro. Inoltre, i dati in formato non valido non devono essere salvati nel database o causare il malfunzionamento di un componente.

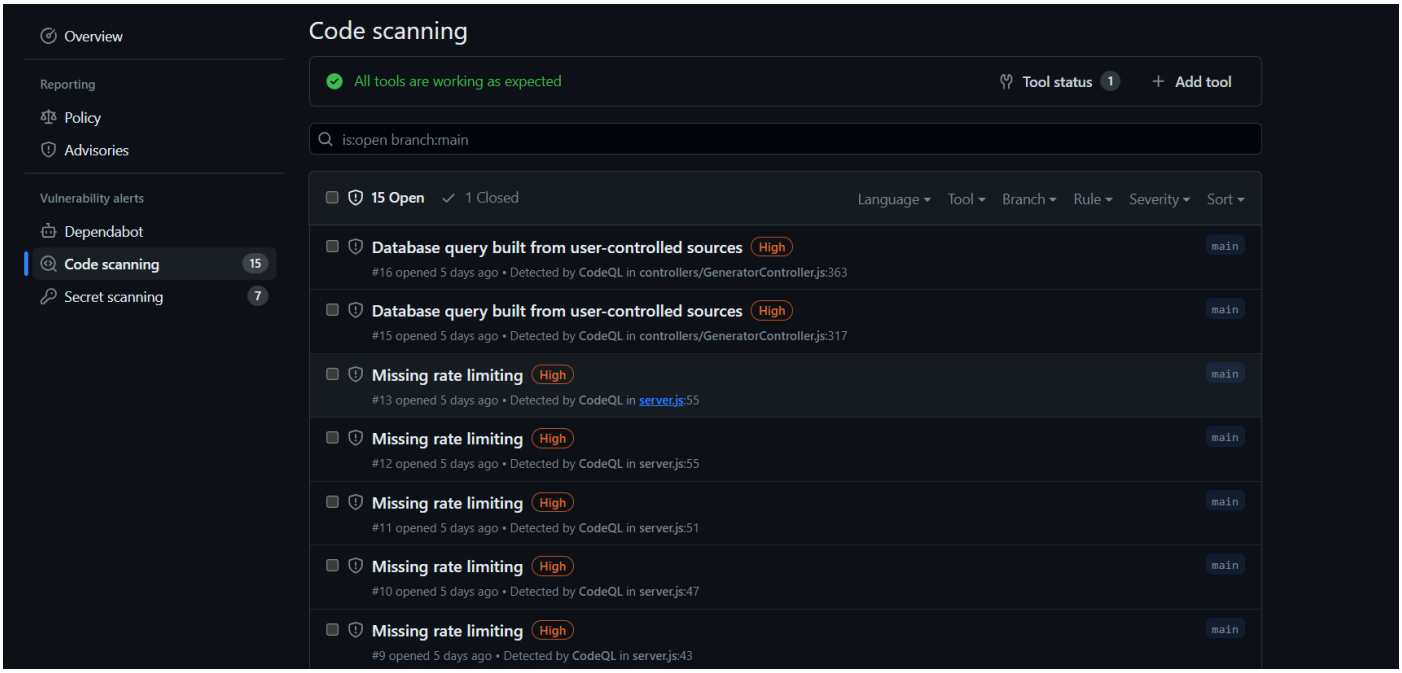
Questo processo di verifica viene affidato ad un middlware di autenticazione di Node.js.

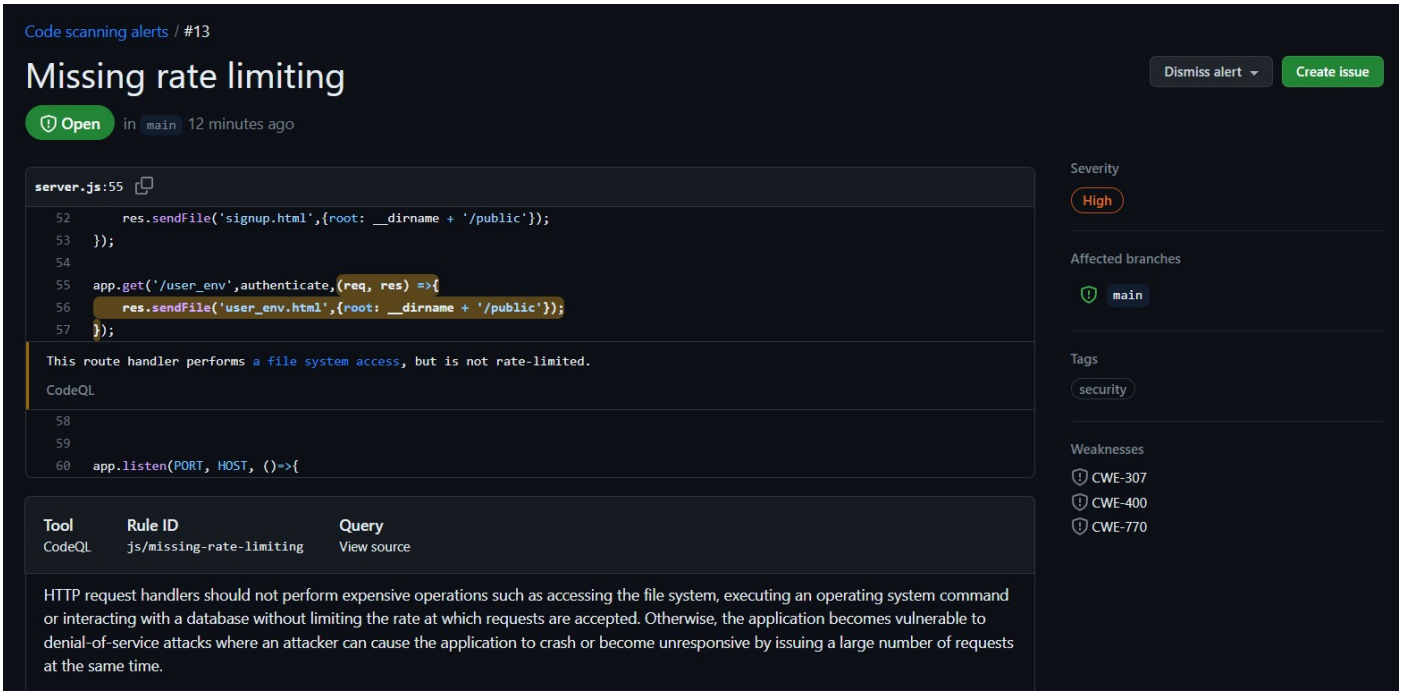
3.3.11 REVISIONE DEL CODICE STATICO (SAST)

L'analisi codice statico, nota anche come analisi del codice sorgente viene in genere eseguita come parte di una revisione del codice, utile ad aumentare la qualità complessiva del codice e ridurre il rischio di creare bug.

In questa fase utilizzeremo degli strumenti di analisi automatizzata, fornite dagli IDE (Visual Studio – Visual Studio Code) e da GitHub Advanced Security, per individuare potenziali vulnerabilità nel codice non in esecuzione, usando tecniche come il controllo dei dati e l'analisi del flusso di dati.

Inoltre, tramite GitHub Advanced Security, è possibile configurare le regole di ramo per i controlli di stato necessari, ad esempio per imporre che un ramo di funzionalità sia aggiornato con il ramo di base prima di unire qualsiasi nuovo codice. Questa procedura garantisce quindi che il ramo sia sempre stato testato con il codice più recente.

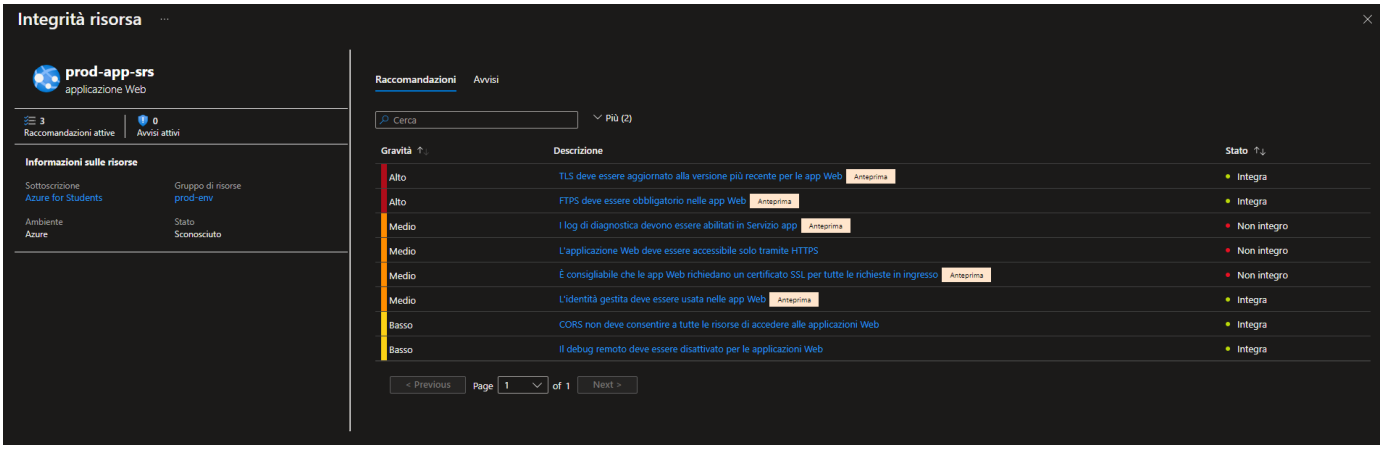




3.3.12 **MONITORAGGIO DELLA SICUREZZA**

Per il monitoraggio della sicurezza abbiamo utilizzato i seguenti strumenti forniti da Azure:

- Microsoft Defender for Cloud: consente di prevenire, rilevare e rispondere alle minacce con maggiore visibilità (e controllo) della sicurezza delle risorse di Azure, incluse le applicazioni Web.



- Monitoraggio di Azure: raccoglie e analizza i dati di telemetria delle app, ad esempio metriche delle prestazioni e log attività. Quando questo servizio identifica condizioni irregolari, avvisa le app e il personale.

3.3.13 **WAF (Web Application Firewall)**

Un altro strumento per evitare attacchi SQL injection e attacchi XXS che abbiamo deciso di adottare è un Web Application Firewall.

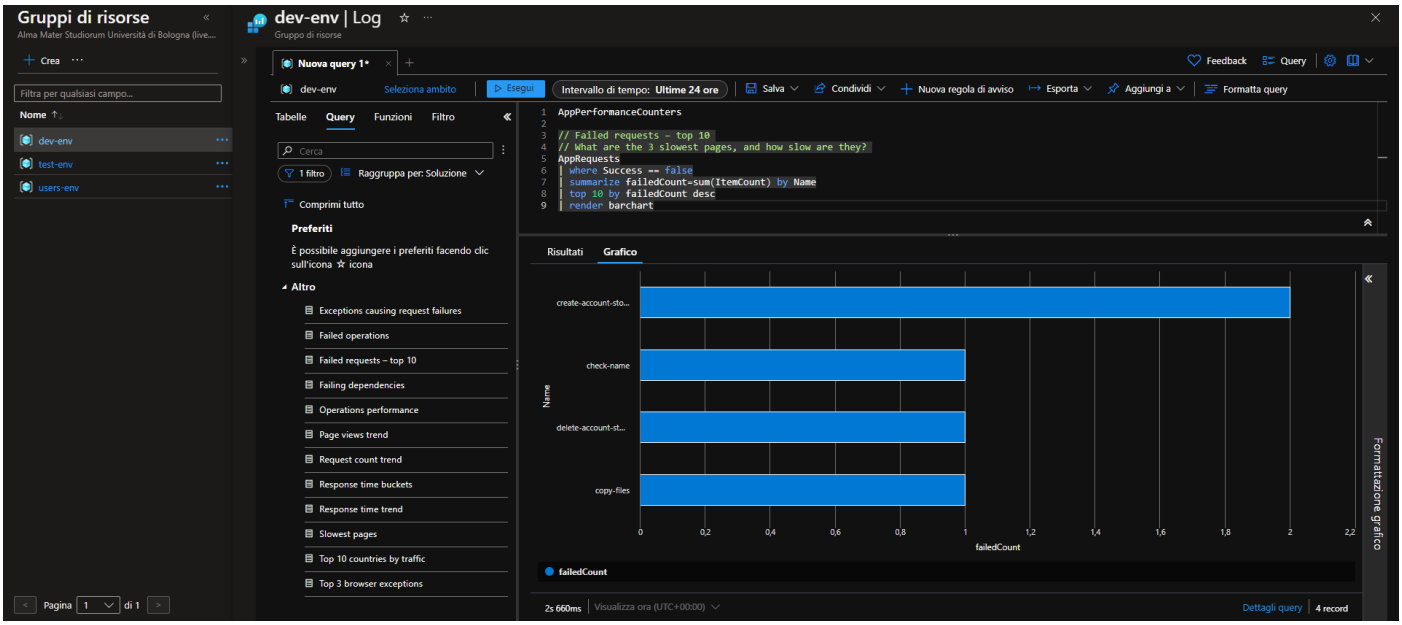
Il WAF di Azure opera sul Gateway dell'applicazione e offre una protezione centralizzata delle applicazioni Web da exploit e vulnerabilità comuni, basandosi su un set di regole derivanti da OWASP.

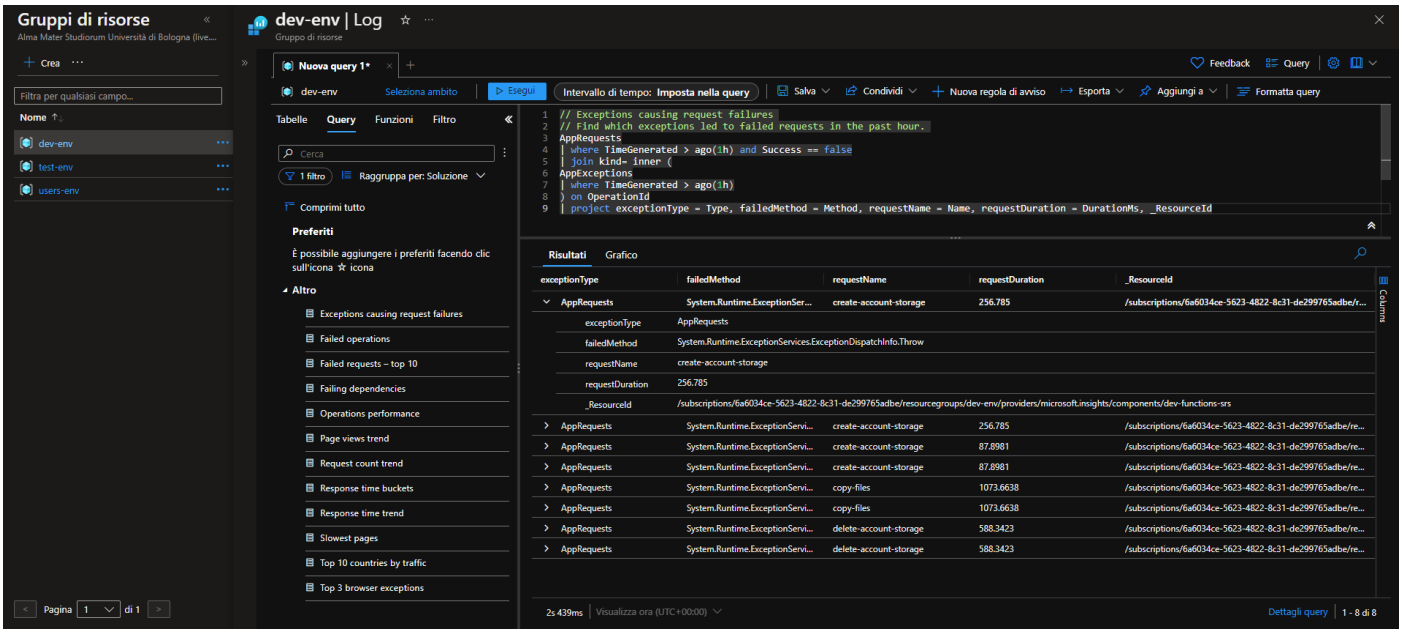
3.3.14 **MONITORAGGIO CENTRALIZZATO DEI LOG**

Per la gestione e il monitoraggio dei log, abbiamo deciso di utilizzare lo strumento Log Analytics di Azure.

Log Analytics oltre ad occuparsi della raccolta dei log, permette di eseguire query di log e analizzare in modo interattivo i risultati.

È quindi possibile usare queste query per supportare altre funzionalità in Monitoraggio di Azure.





3.3.15 **PENETRATION TESTING**

Sempre seguendo il principio di verificare che l'applicazione sia protetta è importante come testare qualsiasi altra funzionalità abbiamo reso i test di penetrazione parte integrante del processo di compilazione e distribuzione, pianificando test di sicurezza e analisi delle vulnerabilità periodici sulle applicazioni distribuite, monitorando eventuali porte aperte, endpoint e attacchi.

La fase di dei test di penetrazione viene affidata a OWASP ZAP, uno strumento open-source che esegue test automatizzati sull'applicazione Web.

Da una prima analisi sono risultate queste vulnerabilità classificate per potenziale rischio e potenziale affidabilità di effettiva esistenza:

Alert counts by risk and confidence

This table shows the number of alerts for each level of risk and confidence included in the report.

(The percentages in brackets represent the count as a percentage of the total number of alerts included in the report, rounded to one decimal place.)

		Confidence				
		User Confirmed	Alto	Medio	Basso	Total
Risk	Alto	0 (0,0%)	0 (0,0%)	0 (0,0%)	0 (0,0%)	0 (0,0%)
	Medio	0 (0,0%)	2 (18,2%)	1 (9,1%)	1 (9,1%)	4 (36,4%)
	Basso	0 (0,0%)	1 (9,1%)	3 (27,3%)	0 (0,0%)	4 (36,4%)
	Informativo	0 (0,0%)	0 (0,0%)	1 (9,1%)	2 (18,2%)	3 (27,3%)
	Total	0 (0,0%)	3 (27,3%)	5 (45,5%)	3 (27,3%)	11 (100%)

Dopo aver deciso una "*soglia di accettazione*" abbiamo cercato di correggere le vulnerabilità partendo da quelle con il rischio e la "*confidence*" più elevati ovvero quelle di grado medio-alto.

4. PIPELINE CI/CD – (Continuous Integration – Continuous Delivery)

La pipeline CI/CD è fondamentale per sfruttare al massimo tutti i benefici offerti dall'approccio DevSecOps.

Per il nostro progetto abbiamo deciso di utilizzare GitHub Actions come strumento per definire la pipeline CI/CD. La scelta di questo tool è dovuta alla sua semplicità di utilizzo e all'integrazione completa sia con i servizi Azure e sia con gli strumenti di sviluppo. Infatti, non appena effettuiamo una push nel main branch del repository su GitHub, si attiva immediatamente la relativa pipeline.

Nello specifico, abbiamo definito tre pipeline dedicate:

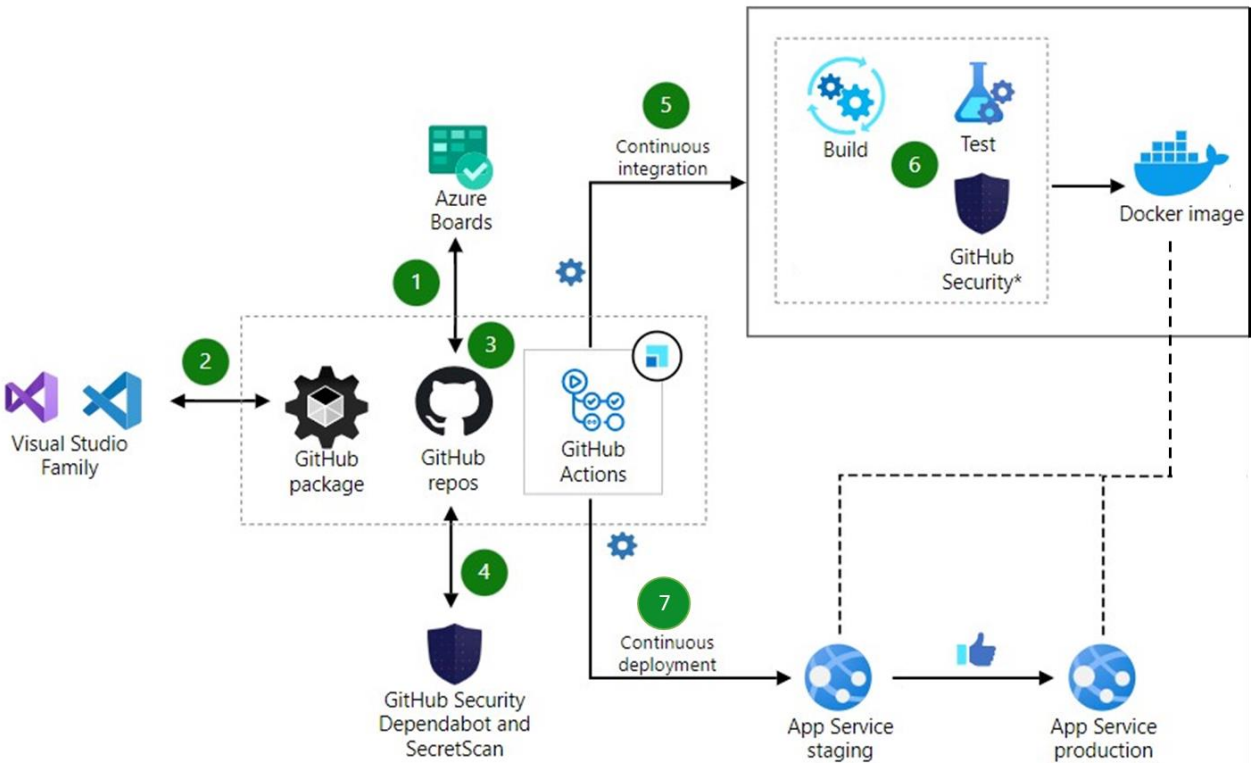
- Infrastructure as Code
- Applicazione Web
- Functions

Ognuna di esse è pensata per automatizzare il controllo della versione del codice, la build, i test nell'ambiente di test, le configurazioni necessarie e la distribuzione nell'ambiente di produzione.

La piena automazione ci consente di distribuire blocchi di codice più piccoli ma più frequentemente.

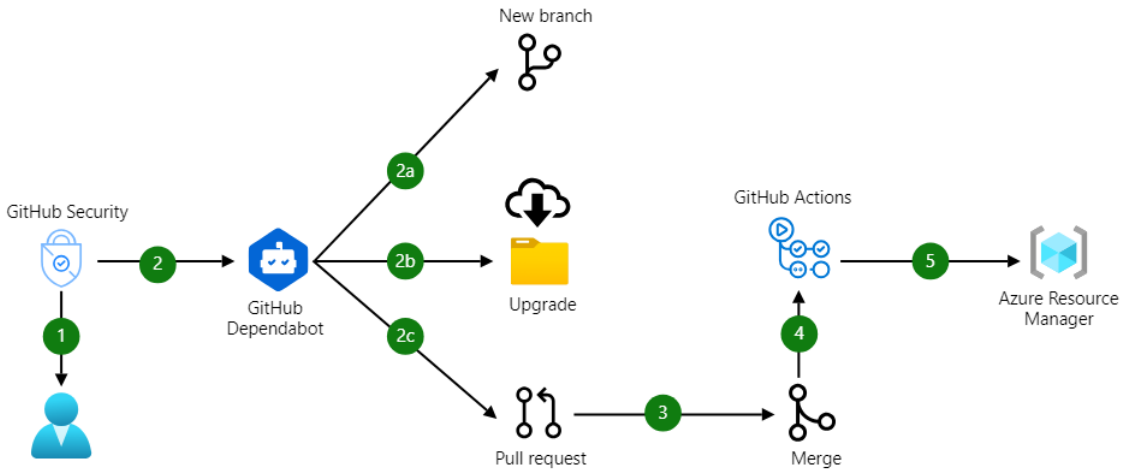
Un'altra caratteristica importante delle pipeline è quella di includere test automatizzati all'interno dell'ambiente di produzione. Questa tipologia di test ci consente di verificare i comportamenti del software in un contesto più realistico e dinamico.

Inoltre, abbiamo scelto GitHub Actions perché ci consente di integrare controllo e sicurezza nel ciclo di Continuous Integration e di Continuous Delivery.



1. Tramite Azure Board gli sviluppatori e i collaboratori controllano gli elementi assegnati e le attività da svolgere.
2. Lo sviluppatore controlla il ramo pertinente all'attività da svolgere sull'ambiente di sviluppo integrato (Visual Studio – Visual Studio Code).

3. Il codice viene archiviato e modificato in un repository GitHub. I rami del repository seguono una strategia di branching specifica, partendo dal ramo principale, sviluppandosi in sequenza.
    a. Ogni versione è associata a un ramo nelle cartelle "Release". Il ramo "main" cambia nel tempo, in base alla versione in produzione in quel momento.
    b. I rami vengono uniti partendo dalle versioni precedenti fino a quelle più recenti, per assicurarsi che qualsiasi correzione di bug di funzionalità o di produzione sia disponibile nelle versioni successive.
4. GitHub Advanced Security analizza tutti i rami, cercando segreti "hardcoded" e vulnerabilità di dipendenza. Se viene rilevata una vulnerabilità, ad esempio una dipendenza da un pacchetto non sicuro deprecato, GitHub invia avvisi ai proprietari e ai gestori dell'organizzazione o del repository. GitHub Dependabot crea un nuovo ramo nel repository, aggiorna automaticamente la dipendenza precedente a una versione sicura e crea una richiesta pull con la dipendenza aggiornata.

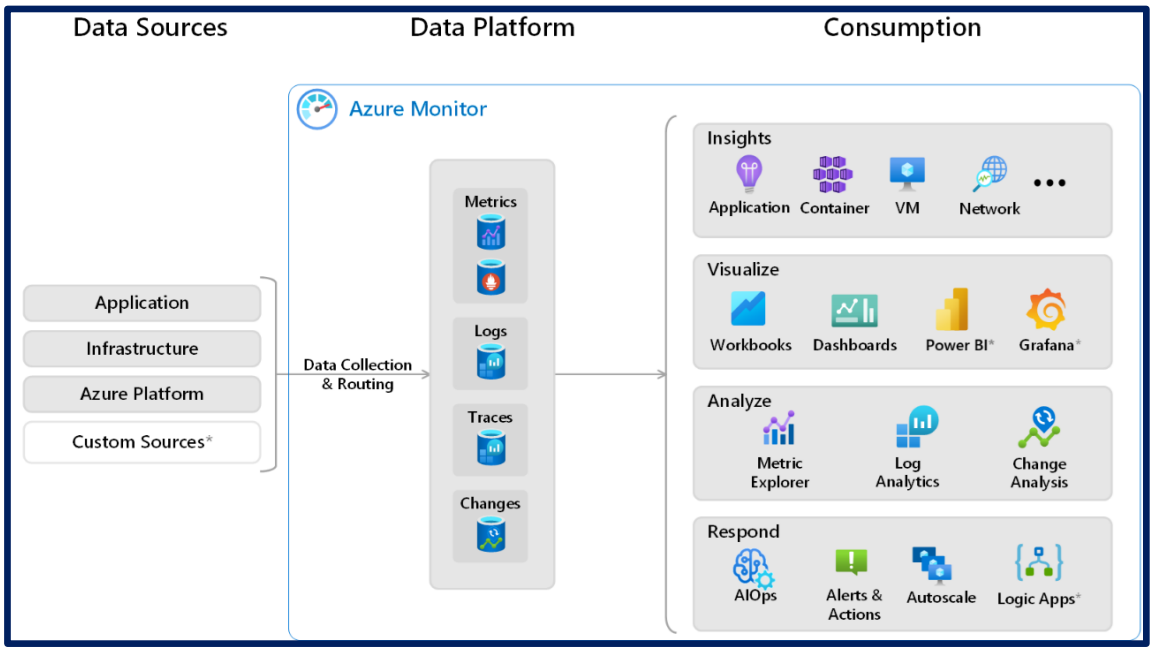


5. L'integrazione continua (CI) viene attivata quando viene creata una nuova richiesta pull oppure il codice viene unito in un ramo di rilascio.
6. Durante lo sviluppo di nuove funzionalità, se vi è una richiesta aperta destinata al ramo di funzionalità o di rilascio, attiva la compilazione, l'esecuzione di test, l'analisi della qualità del codice e la ricerca di vulnerabilità (tramite GitHub Advanced Security).
7. Le pipeline di distribuzione continua (CD) sono in ascolto del caricamento di nuovi contenitori in Azure App Service. Quando è disponibile un nuovo contenitore, la pipeline ne esegue l'analisi, la distribuisce automaticamente nell'ambiente di sviluppo ed esegue test sul nuovo servizio Web.

5. MONITORAGGIO

La distribuzione del prodotto nell'ambiente finale potrebbe sembrare l'ultimo step. In realtà, da questo punto in poi inizia la fase di monitoraggio. Quest'ultima ha come obiettivo monitorare diverse tipologie di dati provenienti da varie sorgenti dati, per far sì che si possano apprendere maggiori informazioni sul funzionamento dell'applicazione e, in generale, del sistema. Queste informazioni possono essere usate per capire i pattern di utilizzo dell'applicazione, la presenza di punti vulnerabili, le performance, la scalabilità, i costi e tanto altro.

Per il monitoraggio usiamo Azure Monitor, che ci fornisce diverse estensioni per monitorare gli aspetti più interessanti del nostro sistema.



Un esempio di monitoraggio che possiamo eseguire è quello per valutare lo stato delle differenti tipologie di risorse che usiamo all'interno del nostro progetto:

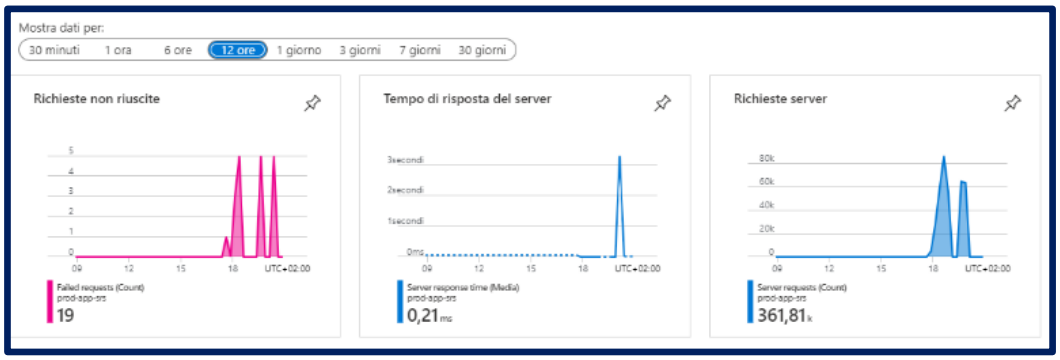
Account di storage

Sottoscrizione	Transazioni	Sequenza temporale transazioni	Latenza end-to-e...	Latenza server	ClientOtherError...	Authentic
Azure for Students (1)	31,8 K					
devstoragesrs	31,8 K		4,3 ms	4,18 ms	51	
Dipartimento di Informatica - S	16,3 K					
prodstoragesrs	16,3 K		4,92 ms	4,62 ms	92	
Azure for Students (1)	31,9 K					
teststoragesrs	31,9 K		5,26 ms	5,21 ms	56	88

Cosmos DB

Sottoscrizione/Database/Raccolta	Unità richiesta	Richieste	Sequenza temporale richieste	Documenti	Utilizzo dei dati
Azure for Students (1)	1055.235	411		9,5	0 B
dev-app-srs-server (2)	1055.235	411		9,5	0 B
Dipartimento di Informatica - S	178.000	60		1	0 B
prod-cosmos-account-srs	178.000	60		1	0 B
Azure for Students (1)	71.633	32		2,8	0 B
test-app-srs-server (2)	71.633	32		2,8	0 B

App web – Ambiente di produzione



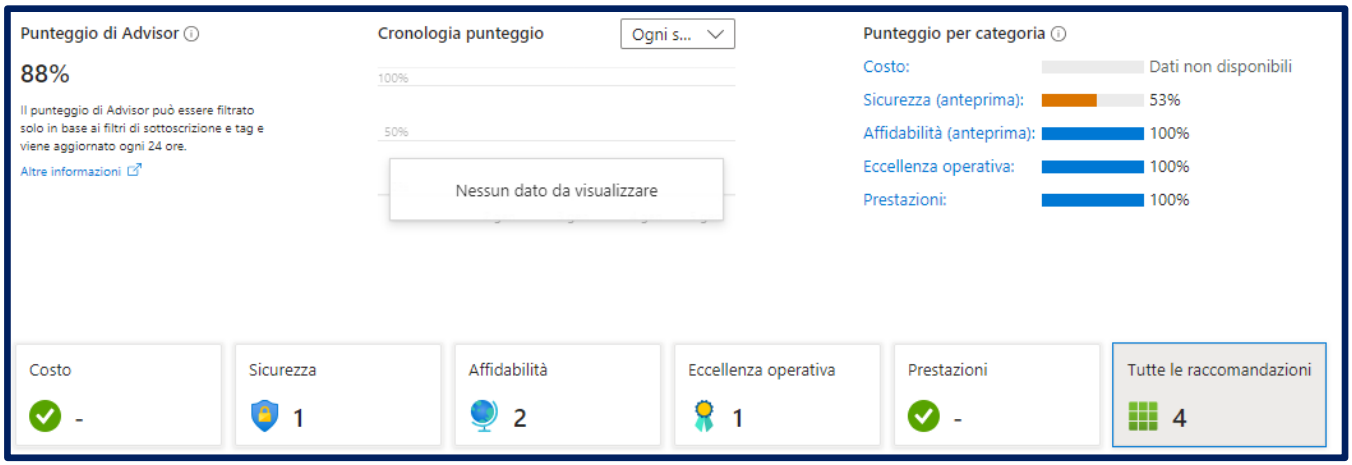
Functions – Ambiente produzione

5.1 MONITORAGGIO – SUPPORTO CON AZURE ADVISOR

Azure Advisor è un servizio di raccomandazioni che analizza la configurazione e l'utilizzo delle risorse e dell'ambiente per proporre dei consigli utili a migliorare la gestione la sicurezza, l'affidabilità, le performance e la gestione dei costi, raggiungendo così l'eccellenza operativa.

Questo strumento ci è stato di grande aiuto per rifinire la configurazione del nostro sistema, visto che non siamo ancora degli esperti e potrebbe esserci sfuggito qualcosa. Inizialmente, il nostro punteggio di Advisor era pari al 88%. Questo significava che c'erano alcuni problemi da sistemare tra cui:

- Sicurezza
    - Definire delle regole di accesso al DB tramite un firewall
- Affidabilità
    - Aggiornare il software del DB dalla versione 4.0 alla 4.2
    - Usare un NAT gateway per le connessioni in uscita
- Eccellenza operativa
    - Migliorare la resilienza abilitando i backup continui su Cosmos DB



Abbiamo seguito i consigli dell'Advisor e sistemato tutte le problematiche individuate.

5.2 MONITORAGGIO – PRESTAZIONE CON APP INSIGHTS

Application Insights è un'estensione che ci consente il monitoraggio delle prestazioni dell'applicazione (Application Performance Monitoring – APM). Gli strumenti APM sono usati per monitorare le applicazioni dallo sviluppo fino alla produzione, focalizzandosi soprattutto sul:

- Capire proattivamente come l'applicazione sta performando.
- Esaminare reattivamente l'esecuzione dell'applicazione per determinare la causa degli incidenti.

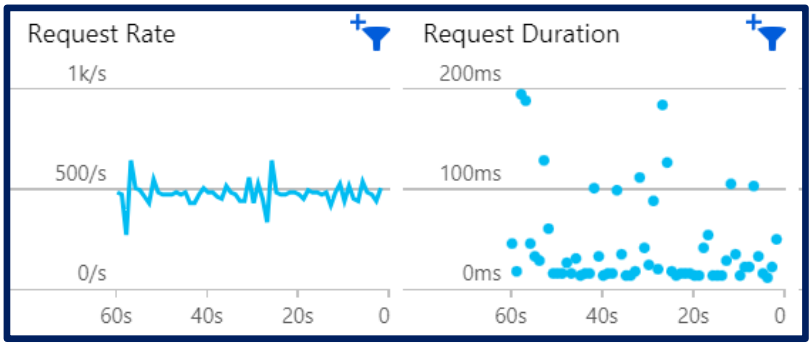
Tutto ciò è possibile grazie alla continua raccolta di dati sulle attività e sullo stato di salute delle varie applicazioni.

Il carico (workload) è uno dei fattori determinanti quando si tratta di valutare le performance di un sistema. Per il nostro progetto, abbiamo simulato l'invio di richieste HTTP da parte degli utenti per testare come reagisse il sistema. Nello specifico, abbiamo eseguito i seguenti tre test:

Test 1: invio di 473 richieste al secondo, simulando 249 utenti virtuali che contemporaneamente e continuamente usano l'applicazione web (singola istanza) per 2 minuti.

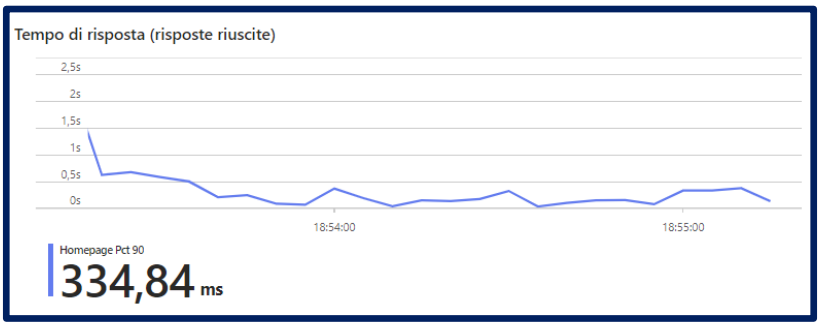
Lato server

Nome server	Richieste
41256ae7ee45	473/sec
CPU usata	**Memoria**
87%	2480 MB



Lato client

Numero totale richieste	Tempo risposta (90° percintile)
56030	347,00 ms

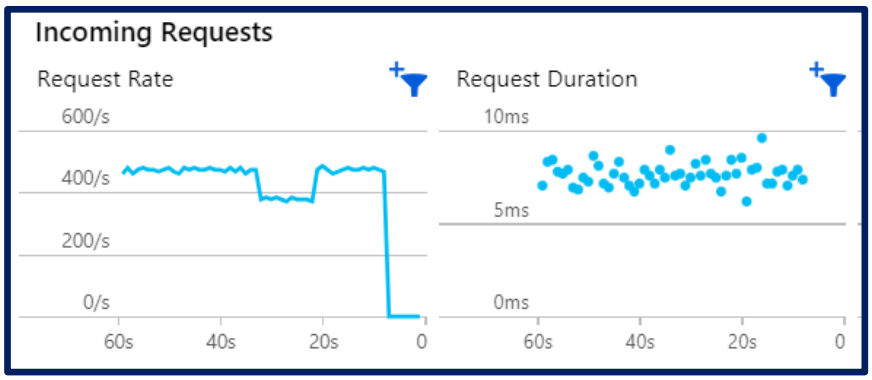


Le performance del primo test potrebbero migliorare con l'aggiunta di nuovi server per l'applicazione web.

Test 2: invio di 473 richieste al secondo, simulando 249 utenti virtuali che contemporaneamente e continuamente usano l'applicazione web (istanze multiple che sfruttano la scalabilità automatica) per 2 minuti.

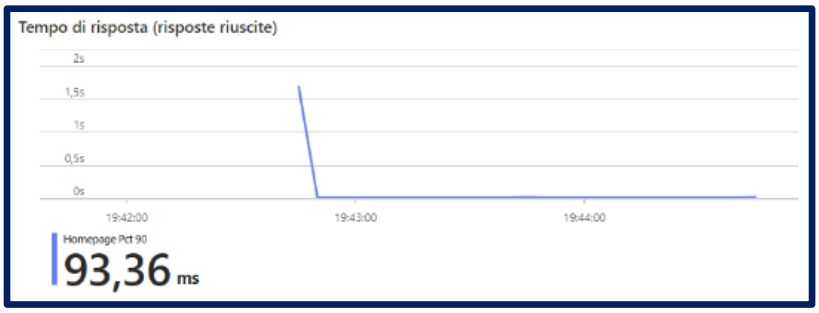
Lato server

Nome server	Richieste	CPU usata	Memoria
0f5cca011b00	94/sec	25%	2120 MB
10e9bcaa28c6	94/sec	23%	2082 MB
a007dc57355b	94/sec	24%	2108 MB
cb2d2bd59bc7	95/sec	22%	2942 MB
cea75a8cfc1d	95/sec	25%	2200 MB



Lato client

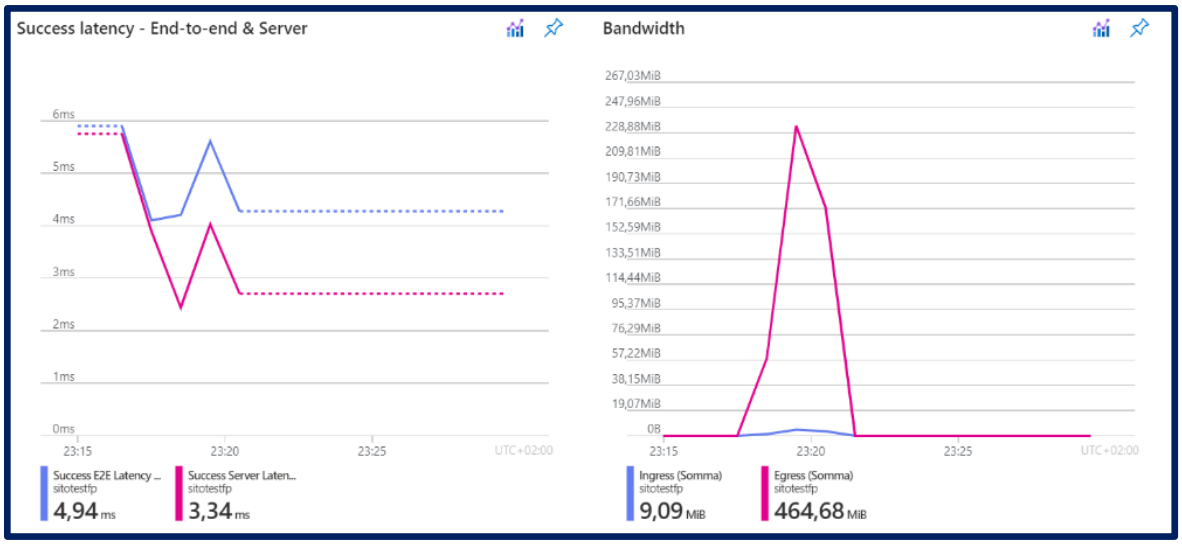
Numero totale richieste	Tempo risposta (90° percintile)
56396	26,00 ms



In questo secondo test l'applicazione web è stata distribuita automaticamente su cinque server, data l'elevata mole di richieste. Dalle analisi emerge che i tempi di risposta sono ampiamente calati e che l'utilizzo delle CPU dei server si attestava mediamente intorno al 25%. Da qui ne consegue che gli utenti che stavano usando l'applicazione non hanno subito rallentamenti e che c'era addirittura potenza di calcolo rimanente per ospitare ulteriori richieste.

Test 3: invio di 473 richieste al secondo, simulando 249 utenti virtuali che contemporaneamente e continuamente usano il sito web di un utente per 2 minuti.

Numero richieste	Latenza Successo E2E	Latenza Successo Server	Disponibilità
55834	4,94 ms	3,34 ms	100%



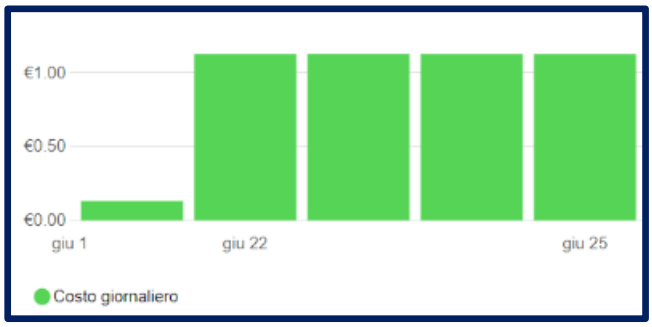
Da questo test emerge che i siti degli utenti garantiscono elevate performance, tempi di risposta ridotti e una disponibilità del 100%.

5.3 MONITORAGGIO – GESTIONE COSTI

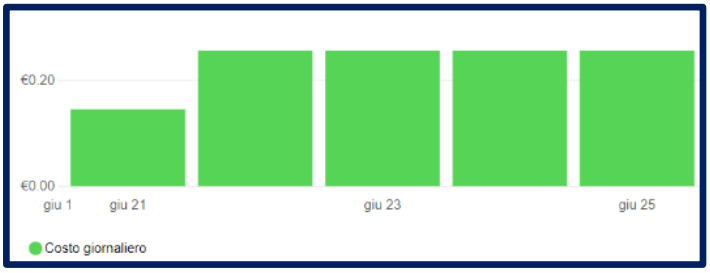
Il monitoraggio dei costi è fondamentale per garantire che il nostro business non fallisca. Per questa tipologia di monitoraggio, Azure mette a disposizione lo strumento Gestione costi che consente di controllare costantemente le spese giornaliere e mensili, divise in base alle risorse e ai servizi che usiamo.

Per il nostro progetto, abbiamo considerato un budget di 250 € al mese. Di conseguenza, le nostre spese dovranno essere calibrate tenendone conto. Per rispettare questo budget, dovremmo spendere circa 8 € al giorno. Dopo un'attenta analisi delle spese, è emerso che spendiamo mediamente circa 5 €/giorno divise tra:

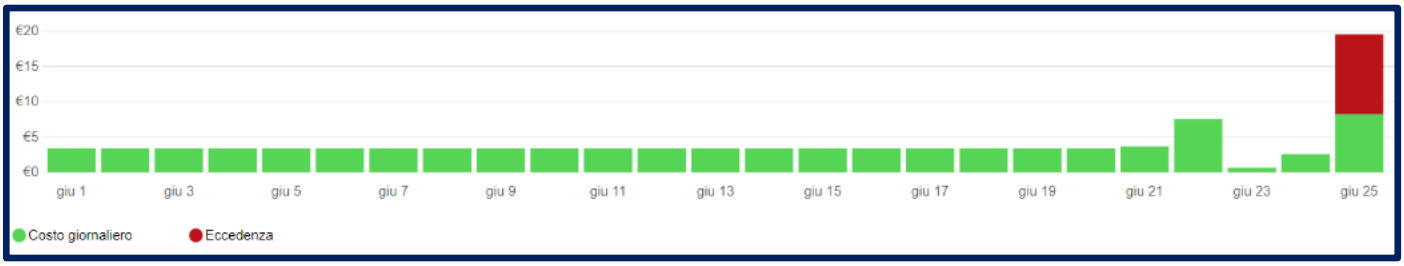
Ambiente di sviluppo: 1,15 €/giorno



Ambiente di test: 0,30 €/giorno



Ambiente di produzione: 3,50 €/giorno



Per quanto riguarda i costi delle principali risorse che usiamo:

- App web e Functions
  - Sviluppo e test: piano Basic B1 da 10,95 €/mese
  - Produzione: piano Premium P1V3 da 109,57 €/mese
- Cosmos DB
  - Gratuito fino a una velocità di 1000 UR/s e 25 GB di spazio utilizzato
  - Dopo, 0,0075 €/ora ogni 100 UR/s in più e 0,234 €/mese ogni GB in più
- Siti web utenti
  - Account di archiviazione con piano Hot
  - 0,02 €/GB per meno di 50 TB di capacità utilizzata
  - 0,0608 € ogni 10000 scritture
  - 0,0047 € ogni 10000 letture
- Test di carico
  - 9,34 €/mese

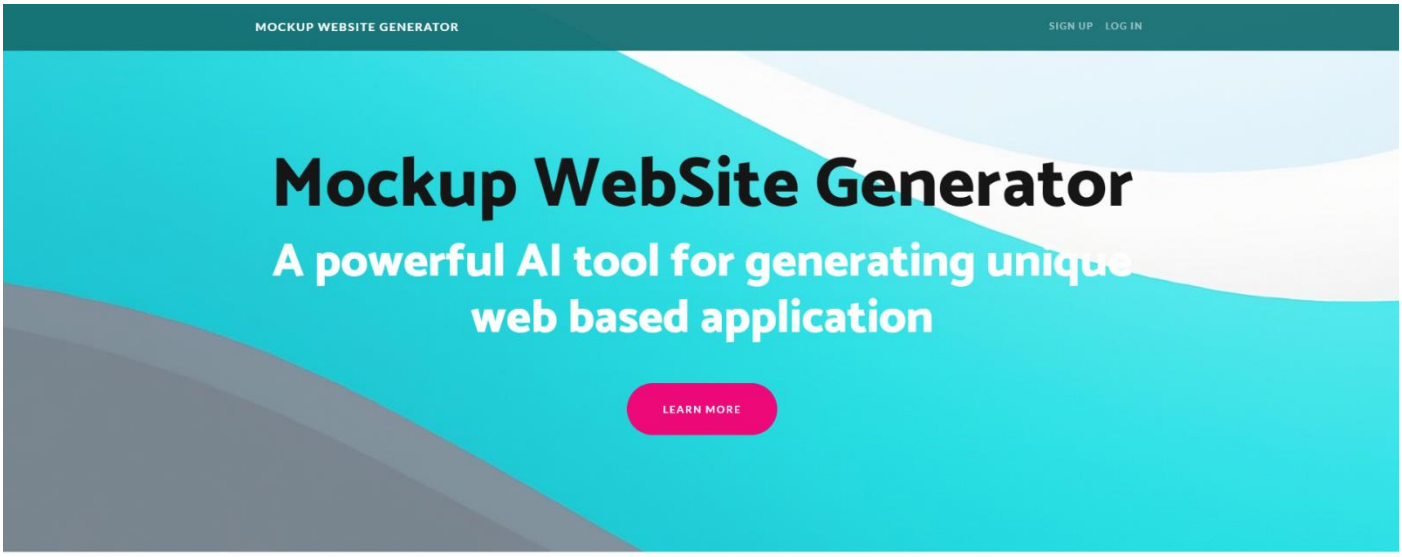
Come è possibile notare dall'elenco soprastante, alcune spese sono fisse e altre variano in base al carico. Al momento il sistema non ha carichi elevati, quindi riusciamo a rientrare nelle spese egregiamente.

Potrebbero però verificarsi dei picchi di carico, come ad esempio avviene nel periodo natalizio, che aumenterebbero le nostre spese repentinamente. Di conseguenza, è importante risparmiare là dove è possibile, in maniera tale da poter destinare una parte del budget a quei momenti in cui le spese potrebbero essere più alte del previsto. Inoltre, abbiamo impostato anche un avviso che segnala quando i costi effettivi superano l'80% del budget mensile.

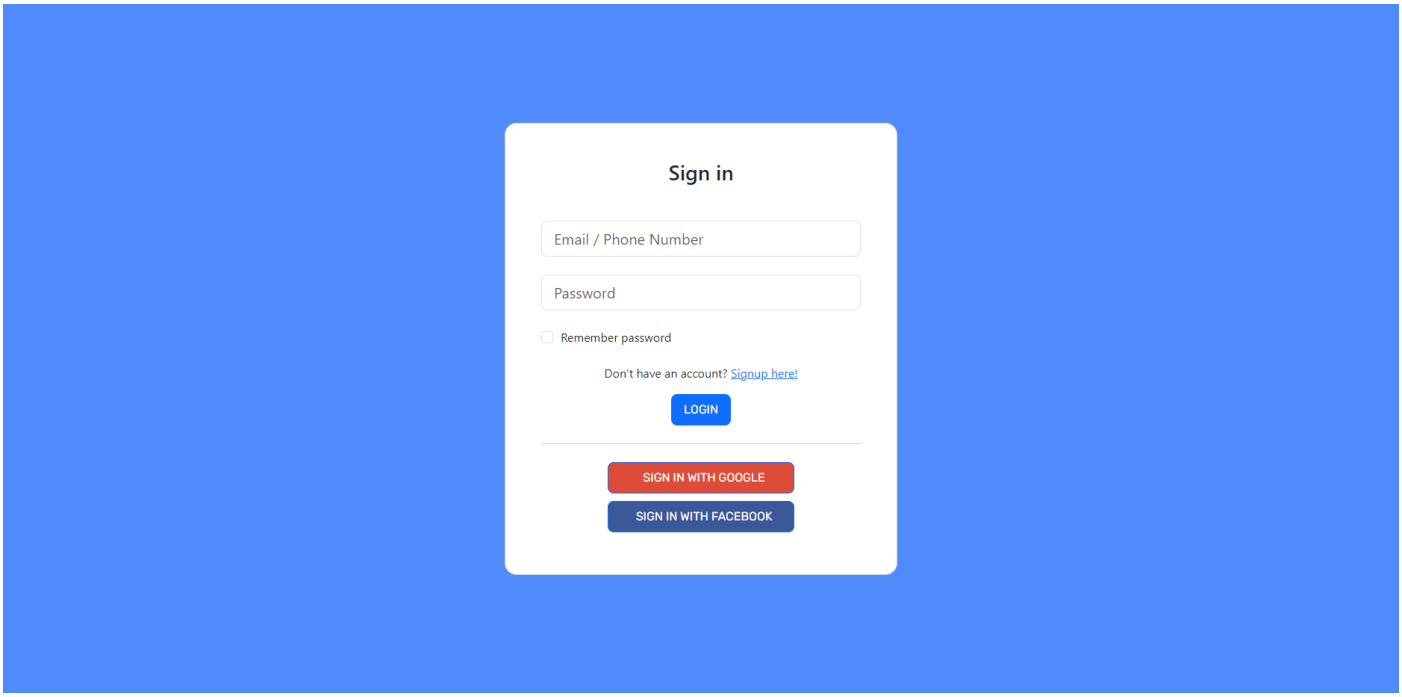
Parlando di risparmi, l'approccio DevSecOps ci permette di risparmiare perché riusciamo a individuare prima e più facilmente gli errori, risolvendoli prima che diventi troppo costoso farlo. Più tardi un errore viene rilevato e più costerà sistemarlo.
Inoltre, potremmo prenotare le risorse, ovvero fare dei contratti di utilizzo per lunghi periodi, ottenendo un risparmio considerevole di circa il 30/40%.

6. FUNZIONAMENTO APPLICAZIONE

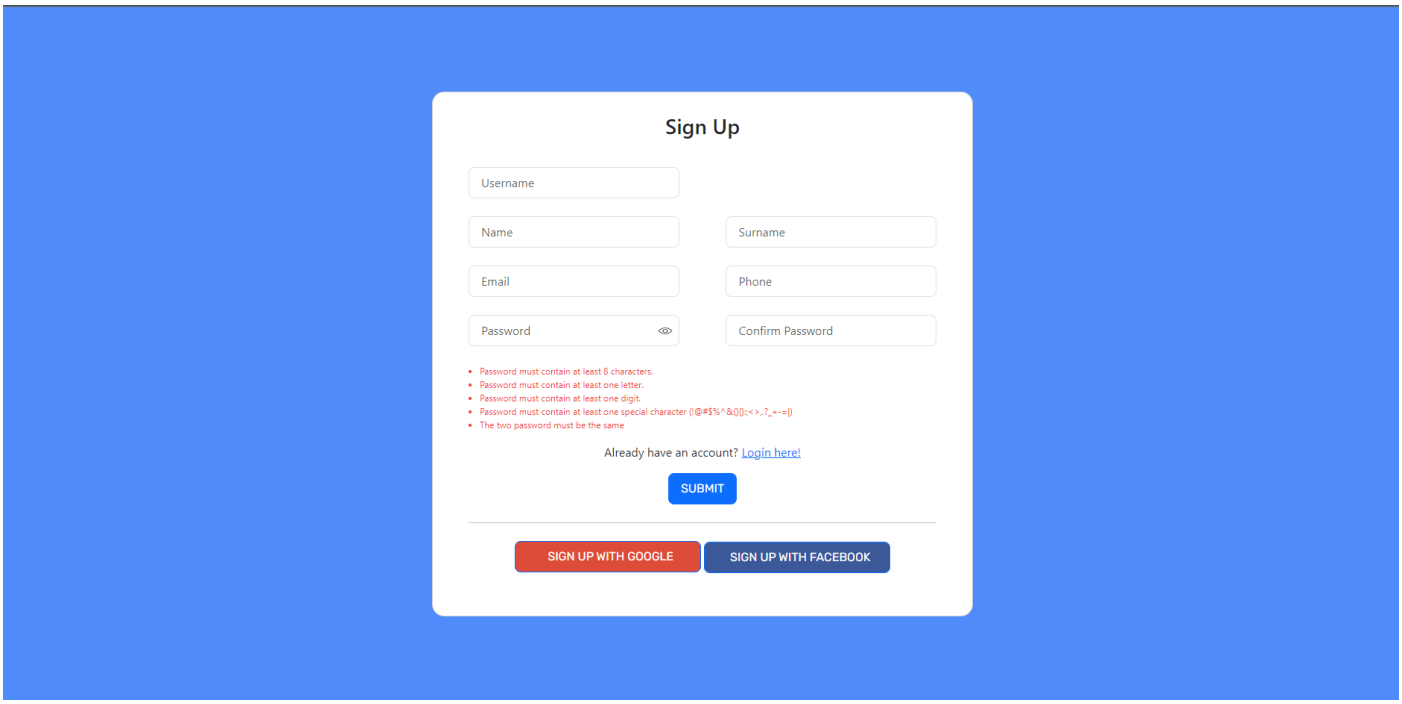
[Homepage]



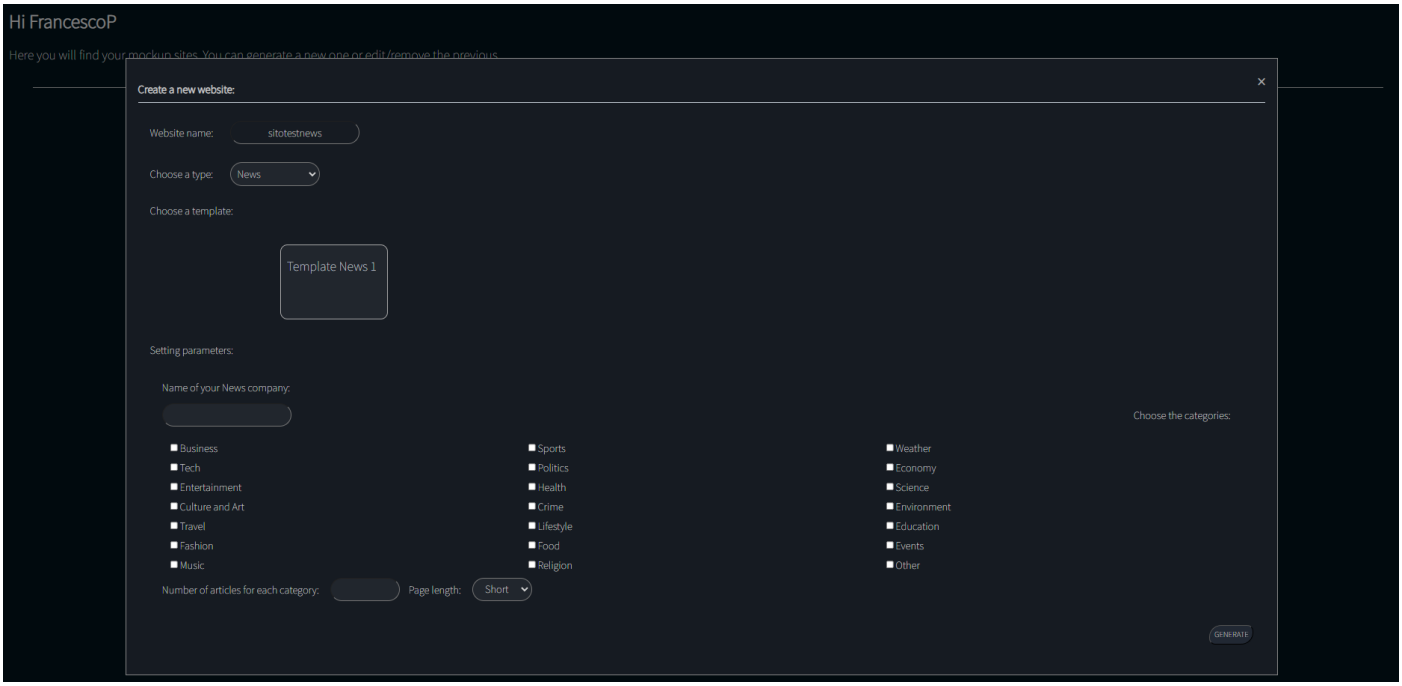
[Sign in]



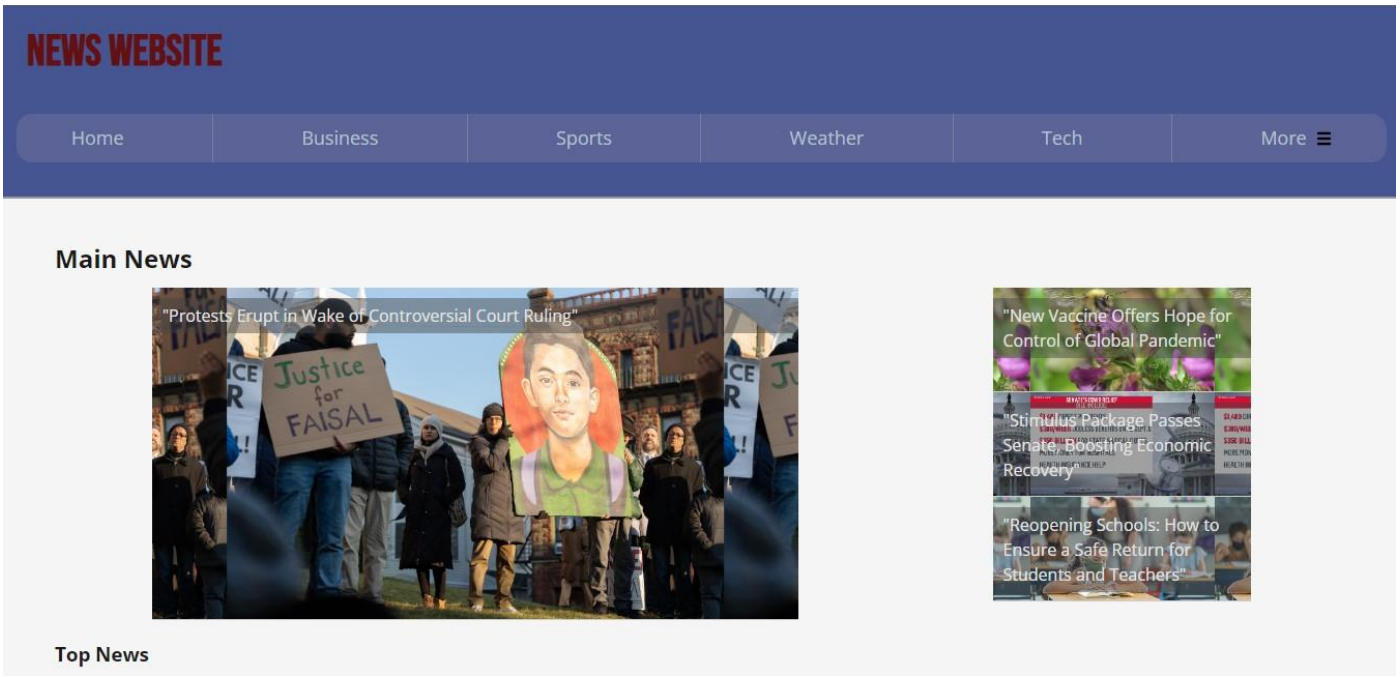
[Sign up]



[Creazione sito]



[Homepage Test News-Website]



[Homepage Test E-commerce Website]