Sistemi Distribuiti

CodeMaster

Alex Testa

Riccardo Rambaldi

Collaborazioni

Il progetto è stato realizzato in collaborazione con i seguenti corsi:

- 69867 - APPLICAZIONI E SERVIZI WEB
- B0011 - SOFTWARE PROCESS ENGINEERING

Che cos'è?

Dal punto di vista di un utente utilizzatore:

- Un applicazione web in cui ogni utente può creare, gestire e risolvere problemi di programmazione in diversi linguaggi, in modo semplice e intutitvo
- Permette a chiunque di mettere alla prova le proprie abilità di programmazione: gli utenti possono creare, organizzare e risolvere sfide di coding, scegliendo tra diversi linguaggi di programmazione e livelli di difficoltà.
- Un sistema, simile al colosso LeetCode, ma con funzionalità gratuite

Che cos'è?

Dal nostro punto di vista:

- Un sistema **distribuito basato su microservizi**
- Un sistema che offre **estensibilità**
- Un sistema che **garantisce isolamento dei guasti**
- Un sistema che **garantisce alta disponibilità**
- Un sistema che **garantisce una scalabilità indipendente**

Dizionario

Nome	**Significato**
Utente	Un cliente abituale che utilizza il sistema
CodeQuest	Si riferisce ad un problema di natura informatica risolvibile attraverso ad un linguaggio di programmazione
Solution	Una soluzione informatica, realizzata mediante un linguaggio di programmazione, relativa ad una codequest

Sistema Distribuito - Definizione

«A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.»

~Leslie Lamport

«Un sistema distribuito è uno in cui il guasto di un computer che non sapevi nemmeno esistere può rendere il tuo computer inutilizzabile.»

Microservizi

Si è scelto l'utilizzo di un sistema basato su microservizi poiché permette:

- **Scalabilità** → Suddividendo la piattaforma in servizi indipendenti, è possibile **scalare solo i componenti più sollecitati** senza influire sugli altri.
- **Resilienza e tolleranza ai guasti** → Se un servizio dovesse fallire, gli altri possono continuare a funzionare (eccetto alcuni casi specifici), **limitando l'impatto dell'errore** sull'intero sistema.
- **Flessibilità ed estendibilità** → Nuove funzionalità possono essere introdotte come **servizi autonomi**, senza modificare il nucleo principale. Ciò rende la piattaforma facilmente **adattabile a nuove esigenze**, come l'aggiunta di nuovi linguaggi di programmazione o strumenti di collaborazione.

Microservizi

Si è scelto l'utilizzo di un sistema basato su microservizi poiché permette:

- **Distribuzione su più nodi** → I microservizi possono essere **distribuiti su diversi nodi o ambienti**, in linea con gli obiettivi di un sistema distribuito e con una migliore **ottimizzazione delle risorse**.
- **Allineamento con il DDD** → I microservizi si integrano perfettamente con il concetto di *bounded context*, permettendo una **mappatura più chiara** tra il modello di dominio e l'architettura software.

Microservizi

Il sistema è suddiviso in più microservizi:

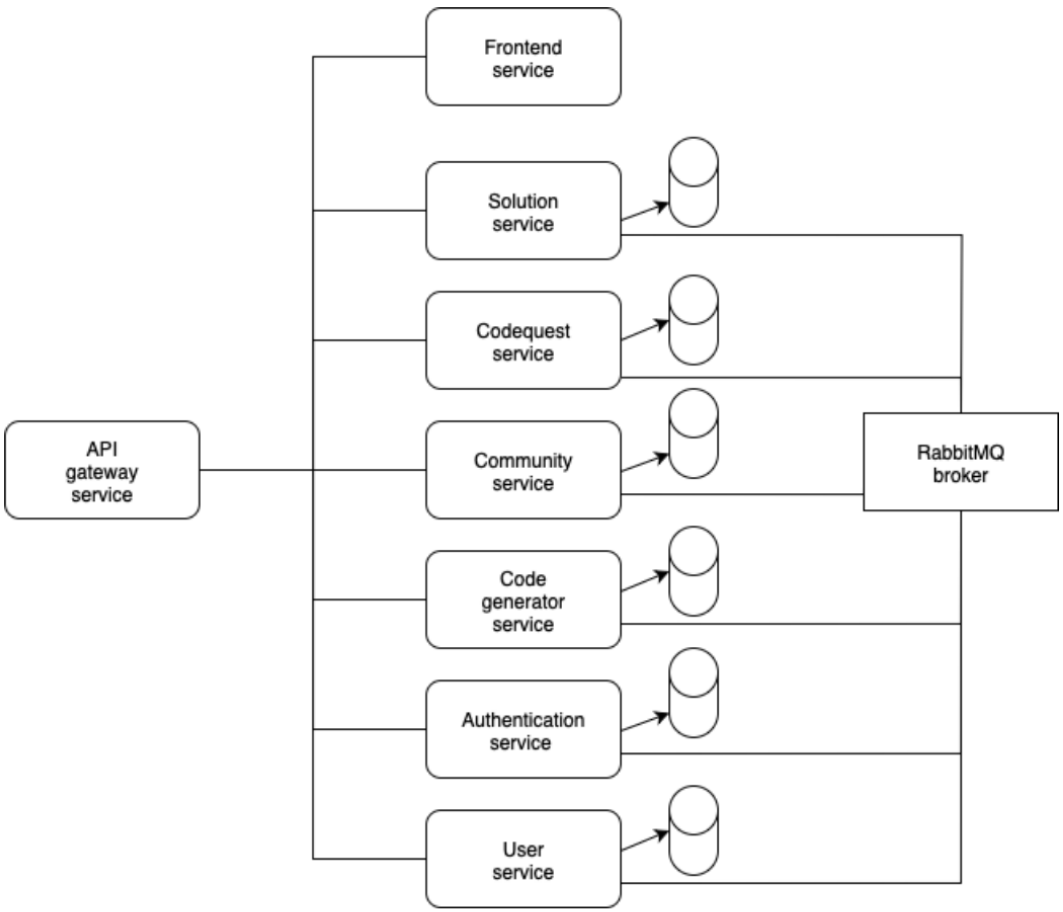
- **AuthenticationService** - Il servizio di autenticazione gestisce il login e il logout degli utenti, verificando le credenziali e mantenendo le informazioni di sessione tramite cookies.
- **UserService** - Il servizio utente gestisce e memorizza le informazioni relative agli account per ogni profilo, come ad esempio, trofei, livelli, cv, bio.
- **CodeQuestService** - Il servizio di codequest gestisce la creazione, l'archiviazione e la manutenzione delle codequest inviate dagli utenti.
- **CommunityService** - Il servizio di community consente agli utenti di aggiungere commenti alle codequest.

Microservizi

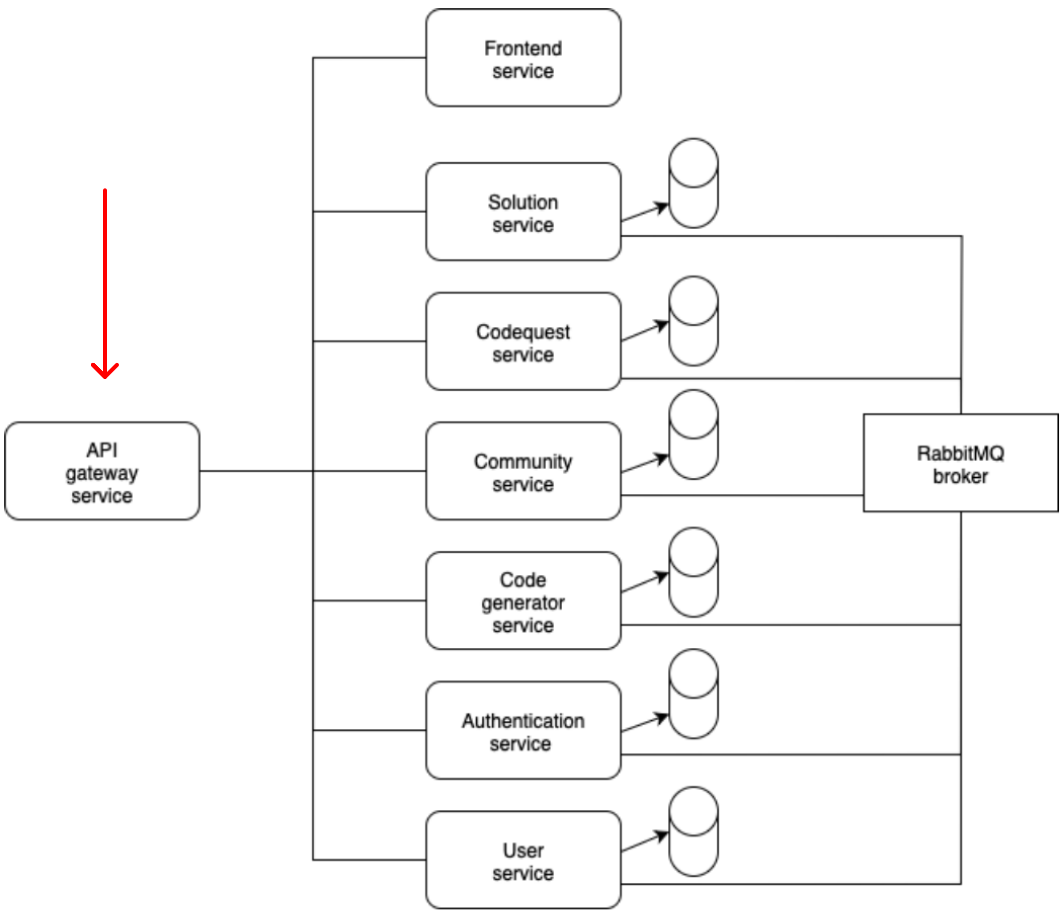
Il sistema è suddiviso in più microservizi:

- **SolutionService** - Il servizio di soluzione gestisce l'archiviazione e la gestione delle soluzioni inviate dagli utenti. Inoltre, esegue il codice utente all'interno di contenitori isolati e dedicati per garantire sicurezza e scalabilità.
- **CodeGeneratorService** - Il servizio di generazione di codice costruisce automaticamente casi di test e modelli di funzione per soluzioni basate su esempi e specifiche forniti dall'utente durante la creazione di un codequest.
- **FrontendService** - Il servizio di frontend fornisce l'interfaccia lato utente e interagisce con tutti i servizi di backend tramite API REST.

Microservizi



API Gateway


Frontend service
Solution service
Codequest service
Community service
Code generator service
Authentication service
User service
API gateway service
RabbitMQ broker

API Gateway

Il sistema include un **reverse proxy** per:

- Separare le richieste esterne al sistema (lato client), dalle richieste interne tra microservizi
- Funge da punto di ingresso per le richieste esterne, indirizzandole ai microservizi appropriati
- Gestisce il bilanciamento del carico
- Gestisce aspetti relativi alla sicurezza delle richieste
- Centralizza le richieste lato utente
- Gestisce una logica di failure per ogni microservizio

Reverse Proxy - NGINIX

- Creato da Igor Sysoev per risolvere problemi legati a carichi elevati e connessioni simultanee
- Funziona bene come reverse proxy e load balancer
- Garantisce anche un servizio di caching
- Ideato per prestazioni elevate e maggiore efficenza rispetto ad un server node

Reverse Proxy

Il sistema include un reverse proxy per:

- Separare le richieste esterne al sistema (lato client), dalle richieste interne tra microservizi
- Funge da punto di ingresso per le richieste esterne, indirizzandole ai microservizi appropriati
- Gestisce il bilanciamento del carico
- Gestisce aspetti relativi alla sicurezza delle richieste
- Centralizza le richieste lato utente
- Gestisce una logica di failure per ogni microservizio

Reverse Proxy - Failure

```
    upstream user_service {
        server codemaster-user-service:4005 max_fails=3 fail_timeout=30s;
    }

    upstream authentication_service {
        server codemaster-authentication-service:4004 max_fails=3 fail_timeout=30s;
    }

    upstream codequest_service {
        server codemaster-codequest-service:3000 max_fails=3 fail_timeout=30s;
    }

    upstream solution_service {
        server codemaster-solution-service:4006 max_fails=3 fail_timeout=30s;
    }

    upstream community_service {
        server codemaster-community-service:4007 max_fails=3 fail_timeout=30s;
    }

    upstream generator_service {
        server codemaster-generator-service:4008 max_fails=3 fail_timeout=30s;
    }
```

Reverse Proxy - Failure

La logica di failure prevede che:

- Se un microservizio fallisce per un massimo di 3 volte consecutive, viene considerato non accessibile
- Ngnix aspetterà 30'' prima di provare a contattare il microservizio
- Se il microservizio è pronto all'utilizzo e ha risolto la failure, allora Nginix tornerà a considerarlo
- Altrimenti Nginix aspetterà altri 30'' per poi riprovare

Reverse Proxy - Sicurezza

La logica di failure prevede che:

- Se un microservizio fallisce per un massimo di 3 volte consecutive, viene considerato non accessibile
- Ngnix aspetterà 30'' prima di provare a contattare il microservizio
- Se il microservizio è pronto all'utilizzo e ha risolto la failure, allora Nginix tornerà a considerarlo
- Altrimenti Nginix aspetterà altri 30'' per poi riprovare

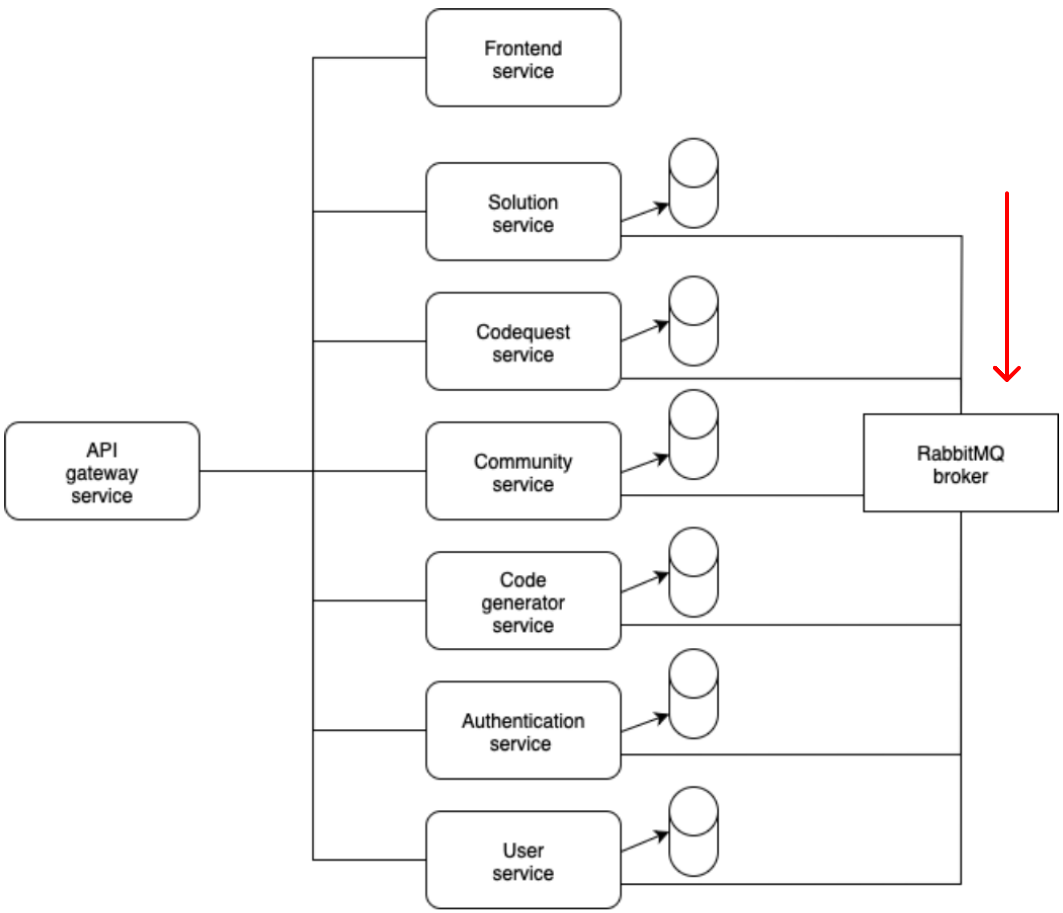
Reverse Proxy - Load Balancing

Nginix permette di:

- Gestire un maggior numero di richieste rispetto ad un server node
- Distribuire uniformemente le richieste dei client in arrivo tra più server beckend
- Considerando un picco di richieste, senza reverse proxy, potrebbe portare il sistema a peccare di disponibilità

Fonte disponibile a questo link

RabbitMQ


Frontend service
Solution service
Codequest service
Community service
Code generator service
Authentication service
User service
API gateway service
RabbitMQ broker

RabbitMQ

RabbitMQ è un **message broker open source** progettato per gestire la comunicazione tra diversi servizi o componenti di un sistema distribuito.

In un'architettura a **microservizi**, svolge il ruolo di **intermediario affidabile** che riceve, conserva e inoltra i messaggi tra i vari servizi, garantendo che nessuna richiesta venga persa, anche in caso di errori o interruzioni temporanee.

RabbitMQ

In un’architettura come quella di **CodeMaster**, dove diversi microservizi devono cooperare (ad esempio per la gestione utenti, la valutazione del codice, o il tracciamento delle attività), RabbitMQ funge da **canale di comunicazione asincrono** tra i componenti.

Ogni microservizio invia i propri messaggi a una **coda (queue)** gestita da RabbitMQ, e gli altri servizi interessati li **consumano** quando sono pronti.

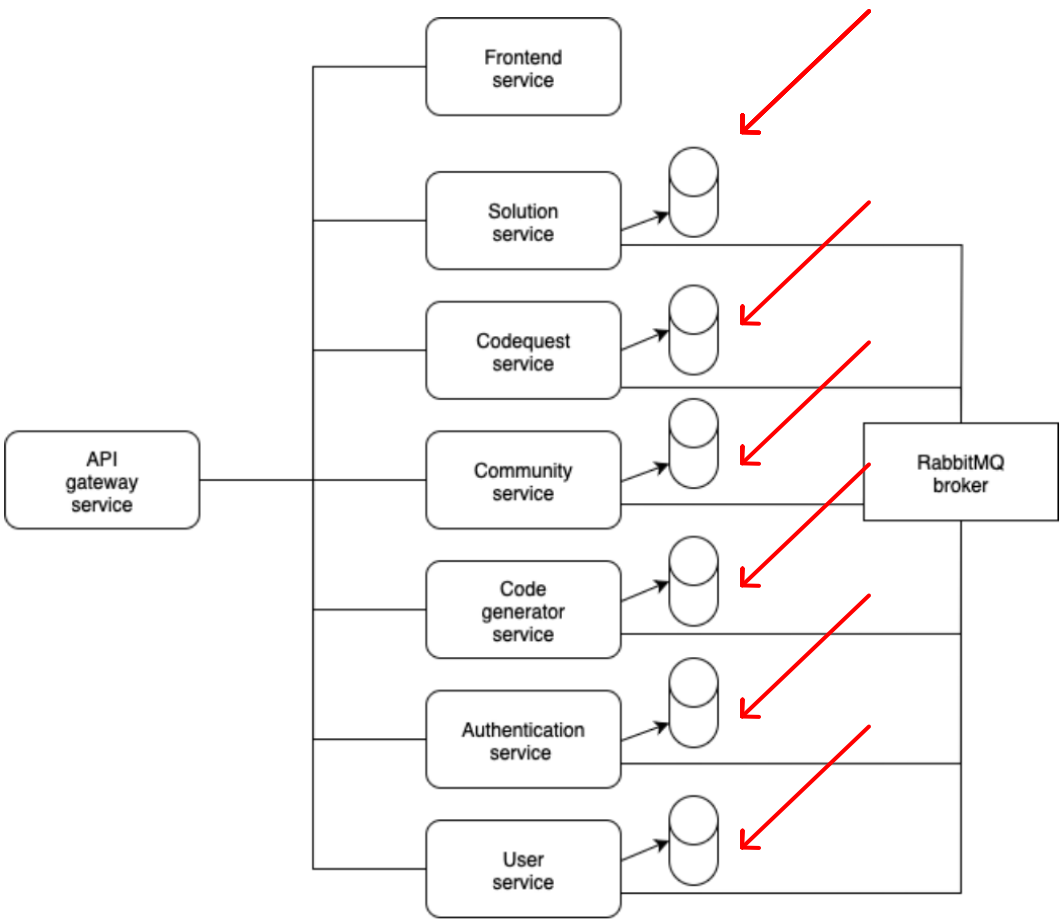
Questo approccio permette di **disaccoppiare i servizi**, evitando dipendenze dirette e migliorando la **resilienza complessiva del sistema**.

RabbitMQ

In sintesi RabbitMQ garantisce:

- **Affidabilità e tolleranza ai guasti** - I messaggi vengono memorizzati in **code persistenti**, che rimangono attive anche in caso di crash o riavvio di un servizio.
- **Comunicazione asincrona e non bloccante** - I servizi possono inviare messaggi senza attendere una risposta immediata, migliorando la **reattività e la scalabilità** del sistema.
- **Gestione ordinata e controllata del carico** - Le code fungono da **buffer**, bilanciando automaticamente il flusso di richieste ed evitando sovraccarichi improvvisi nei servizi backend.
- **Scalabilità e flessibilità architetturale** - consente di adattare facilmente la comunicazione alle esigenze di crescita e complessità del sistema.

MongoDB


Frontend service
Solution service
Codequest service
Community service
Code generator service
Authentication service
User service
API gateway service
RabbitMQ broker

MongoDB

MongoDB è un **database NoSQL orientato ai documenti**, progettato per offrire **flessibilità, scalabilità e alte prestazioni**.

A differenza dei database relazionali tradizionali, MongoDB archivia i dati in **documenti JSON (BSON)**, una struttura dinamica che permette di gestire in modo naturale informazioni eterogenee e in continua evoluzione.

In un'architettura come quella di **CodeMaster**, ogni microservizio mantiene un proprio **database indipendente** basato su MongoDB.

MongoDB

Questo approccio riflette il principio del **database per servizio**, dove ciascun componente gestisce i propri dati in autonomia, evitando dipendenze dirette tra i servizi e garantendo un **elevato livello di isolamento e coerenza**.

MongoDB

In sintesi:

- **Flessibilità nello schema dei dati** - I documenti BSON permettono di aggiungere o modificare campi senza alterare la struttura complessiva del database
- **Scalabilità orizzontale (sharding)** - MongoDB può distribuire i dati su più nodi, garantendo prestazioni elevate anche con grandi volumi di informazioni o carichi distribuiti
- **Alta disponibilità e replica automatica** - Tramite i **replica set**, MongoDB assicura la persistenza dei dati e la tolleranza ai guasti, mantenendo copie sincronizzate su più server
- **Perfetta integrazione con i microservizi** - Ogni servizio può mantenere il proprio database locale, semplificando la gestione, migliorando le prestazioni e permettendo una maggiore indipendenza nella scalabilità

Fault Tolerance

La tolleranza ai guasti nell'architettura proposta è ottenuta attraverso una combinazione di :

- Timeout
- Retry Mechanisms (spiegato precedentemente)
- Strategie di gestione degli errori
- Monitoraggio in tempo reale

Fault Tolerance - Timeout e Gestione di errori

- Tutte le richieste API dal frontend ai servizi backend sono configurate con un **timeout di 5 secondi**.
- Ciò garantisce che, nel caso in cui un servizio diventi non disponibile, la richiesta fallisca in modo corretto invece di lasciare il client in uno stato di blocco
- Questa strategia consente alla piattaforma di rimanere parzialmente operativa anche se alcuni servizi di backend sono temporaneamente inattivi
- Ad esempio, se il servizio utente non è disponibile, non è possibile recuperare i dati del profilo dell'utente,
  ma l'utente è comunque in grado di creare e risolvere codequest.

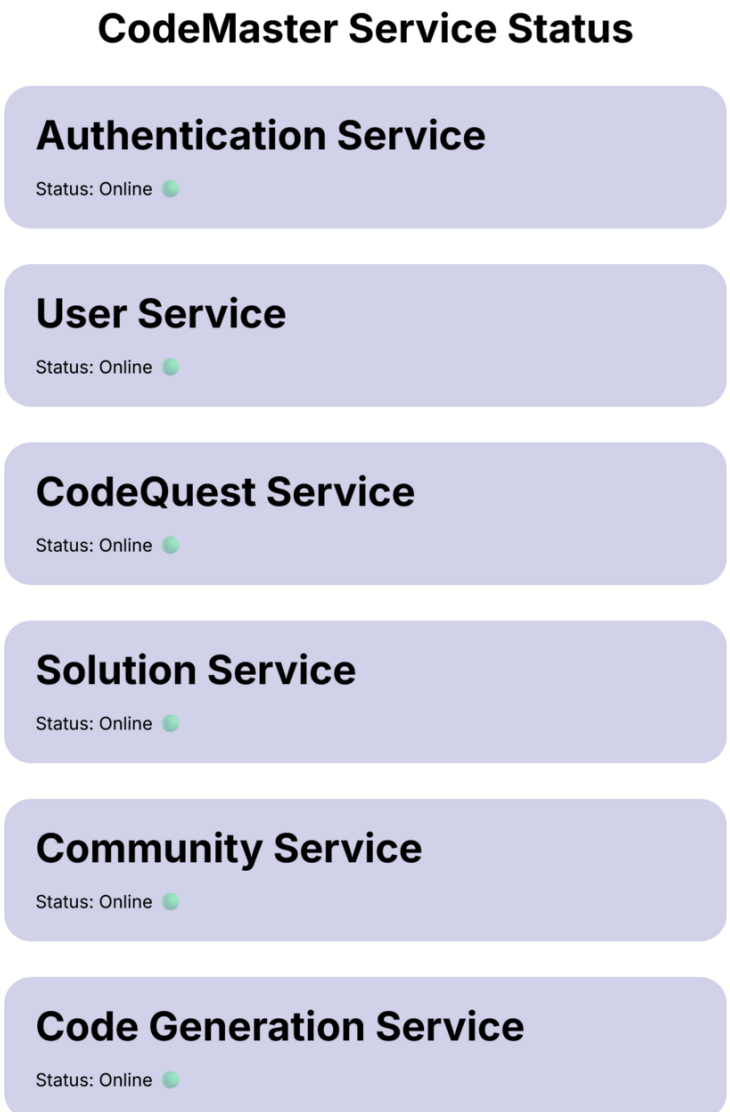
Fault Tolerance - Timeout e Gestione di errori

- Questo viene reso possibile grazie ai cookies, con i relativi problemi alla sicurezza.
- Ma funziona solo dopo essersi autenticati.
- Infatti se l'utente non riesce ad autenticarsi, non potrà accedere all'applicativo
- L'**authentication service** viene definiro **sbarrante** per il nostro sistema

Fault Tolerance - Monitoraggio

- All'interno del frontend è disponibile una dashboard dedicata che fornisce informazioni in tempo reale sullo stato di salute di tutti i microservizi.
- Questa funzionalità di monitoraggio consente agli utenti e agli amministratori di rilevare immediatamente i guasti, verificare la disponibilità dei servizi e intraprendere azioni correttive quando necessario.
- Questa dashboard contribuisce al rilevamento proattivo dei guasti e all'affidabilità operativa.

Fault Tolerance - Monitoraggio



HealthCheck e Avvio

Dato che il sistema è dockerizzato:

- Ogni microservizio ha una rotta /status per controllare l'effettivo funzionamento del servizio
- Questa logica è utilie sia per capire quali servizi sono disponibili e quali no
- Sia per capire quali container sono effettivamente pronti ad erogare il servizio
- Di fatto il sistema viene orchestrato tramite un file docker-compose che si occupa anche di gestire gli ordini di avvio dei container
- Ad esempio se un container include rabbit, deve aspettare che rabbit sia effettivamente funzionante prima di potersi avviare

HealthCheck e Avvio

```yaml
rabbitmq:
   image: rabbitmq:4.1.4-management
   ports:
     - '5672:5672'
     - '15672:15672'
   environment:
     RABBITMQ_DEFAULT_USER: guest
     RABBITMQ_DEFAULT_PASS: guest
   volumes:
     - rabbitmq_data:/var/lib/rabbitmq
   restart: always
   healthcheck:
     test: ["CMD", "rabbitmq-diagnostics", "ping"]
     interval: 10s
     timeout: 5s
     retries: 5



 api-gateway:
   build:
     context: ./api-gateway
     dockerfile: Dockerfile
   ports:
     - '80:80'
   depends_on:
     codemaster-user-service:
       condition: service_healthy
     codemaster-authentication-service:
       condition: service_healthy
     codemaster-codequest-service:
       condition: service_started
```

Code Generation

Quando un utente crea una nuova Codequest, sarà richiesta la **signature completa** della funzione che dovrà essere implementata da ogni utente che vorrà risolvere la Codequest, ovvero:

- nome della funzione
- tipo di ritorno della funzione
- quanti parametri in input
- tipo e nome di ogni parametro in input

Oltre a queste informazioni, sono richiesti all'utente anche degli esempi (almeno uno), ossia una serie di combinazioni di valori dei parametri in input e il corrispondente output atteso.

Code Generation - come funziona

L'insieme di queste informazioni vengono ricevute dal Codequest service che le inoltra tramite rabbitMQ al servizio di code generation. Quando quest'ultimo riceve il messaggio, un parser estrae le informazioni e le concatena per generare:

- il **template della funzione** da implementare per risolvere le Codequest
- i **test** (uno per ogni esempio fornito) che saranno usati per testare che la funzione implementata sia corretta

Sia il template che i test sono generati per ogni linguaggio utilizzabile specificato nella Codequest (Scala, Java e Kotlin) e salvati come singolo documento nel repository del servizio.

In questo modo, quando un utente accede ad una Codequest per risolverla, verrà presentato già il template della funzione da implementare.

Esecuzione e test del codice

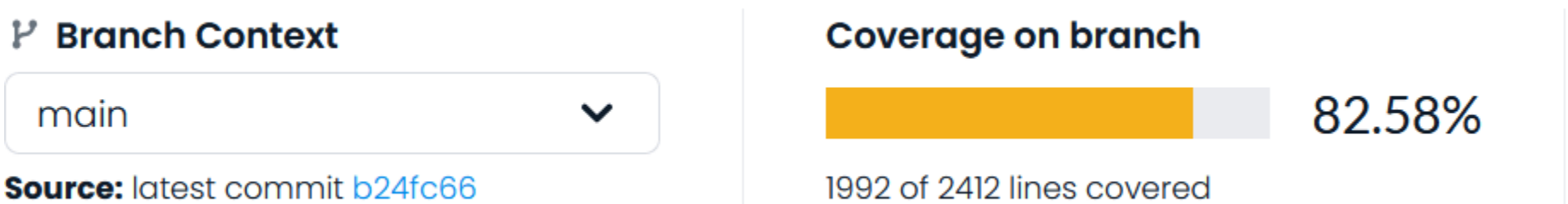
Ogni volta che l'utente esegue debug o submit del codice, all'interno del backend viene eseguito e lanciato un container docker che si occupa di compilare e/o testare la funzione. Quando l'esecuzione/compilazione termina, il container viene eliminato e l'output è parsato per mostrare all'utente:

- se la compilazione è terminata con successo oppure è fallita e nel caso mostrare lo stack trace degli errori
- se i test sono passati con successo oppure quali test sono falliti mostrando l'output atteso e l'output ricevuto (per ogni test)

Test

Il sistema è stato testato tramite:

- **Unit Test** - per la parte relativa al dominio e alla logica di business
- **Integration Test** - per la parte di dialogo con le infrastrutture (MongoDB, Rabbit, ecc..), facendo il mock tramite librerire apposite (es. MongoMemoryServer) e per la parte di interazione tra più moduli del sistema



Deploy

Grazie alla collaborazione con il corso di

SOFTWARE PROCESS ENGINEERING

il deploy avviene tramite:

- Github Actions, che si preoccupa di creare le immagini per ogni microservizio e versionarle con la versione corrente dell'applicativo, ad esempio se CodeMaster è alla versione 1.0.0 tutte le sue immagini sono alla versione 1.0.0
- DockerHub, infatti tutte le immagini create vengono caricate sulla piattaforma e rese disponibili e accessibili a chiunque

Deploy

wealecs/community-service	8 days ago	IMAGE	Public	Inactive
wealecs/authentication-service	8 days ago	IMAGE	Public	Inactive
wealecs/codequest-service	8 days ago	IMAGE	Public	Inactive
wealecs/user-service	8 days ago	IMAGE	Public	Inactive
wealecs/solution-service	8 days ago	IMAGE	Public	Inactive
wealecs/code-generator-service	8 days ago	IMAGE	Public	Inactive
wealecs/multi-lang-runner	8 days ago	IMAGE	Public	Inactive
wealecs/codemaster	6 months ago		Public	Inactive

GRAZIE PER L'ATTENZIONE