Architettura dei sistemi software

Progetto per l'A.A. 2021/2022

20 maggio 2022

Premessa

Il progetto del corso di Architettura dei sistemi software è relativo alla sperimentazione pratica di alcune tecnologie studiate durante il corso, e riguarda la realizzazione di una semplice applicazione a servizi, nonché il rilascio di questa applicazione in un opportuno ambiente di esecuzione.

Il progetto può essere realizzato in diverse varianti, descritte nelle seguenti sezioni.

Progetto (punto di partenza)

Edipogram è un semplice social network per la condivisione di enigmi (ovvero, giochi enigmistici). Il codice di **Edipogram** è disponibile sul repository GitHub del corso (https://github.com/aswroma3/asw/tree/master/projects/asw-edipogram).

Un enigma è un gioco enigmistico, come per esempio il seguente:

Cambio – Cambio di vocale (7)

Ponte pericolante

Saldo non è…
e per questo può essere grave!

(Il Valletto)

Ogni enigma è caratterizzato da:

- un tipo (generico) (nell'esempio, un Cambio)
- un tipo specifico (nell'esempio, un Cambio di vocale (7))
- un titolo (nell'esempio, Ponte pericolante)
- un testo (nell'esempio, *Saldo non è … / e per questo può essere grave!*)
- un autore (nell'esempio, Il Valletto)
- una soluzione (nell'esempio, accOnto/accEnto).

Edipogram viene usato come segue. Gli utenti del sistema possono pubblicare degli enigmi. Possono poi seguire gli enigmi di specifici tipi. Per esempio, Alice potrebbe seguire gli enigmi di tipo Anagramma e quelli di tipo Cambio. Quando un utente accede alla pagina degli enigmi che segue, gli vengono mostrati gli enigmi dei tipi che segue.

L'applicazione **Edipogram** è composta dai seguenti microservizi:

- Il servizio **enigmi** gestisce gli enigmi. Ogni enigma ha un autore, un tipo, un tipo specifico, un titolo, un testo (che può essere composta da più righe) e una soluzione (che può essere composta da più parole). Operazioni:
  - POST /enigmi aggiunge un nuovo enigma (dati autore, tipo, tipo specifico, titolo, testo e soluzione)
  - GET /enigmi/{id} trova un enigma, dato l'id
  - GET /enigmi/{id}/soluzione trova la soluzione di un enigma, dato l'id
  - GET /enigmi trova tutti gli enigmi (senza la soluzione)

    - GET /cercaenigmi/autore/{autore} trova tutti gli enigmi di un certo autore (senza soluzione)
    - GET /cercaenigmi/autori/{elenco-di-autori} trova tutti gli enigmi di un insieme di autori (senza soluzione)
    - GET /cercaenigmi/tipo/{tipo} trova tutti gli enigmi di un certo tipo (senza soluzione)
    - GET /cercaenigmi/tipi/{elenco-di-tipi} trova tutti gli enigmi di un insieme di tipi (senza soluzione)
- Il servizio **connessioni** gestisce le connessioni degli utenti, ovvero i tipi di enigmi seguiti dagli utenti. Le connessioni sono delle coppie utente-tipo. Operazioni:
    - POST /connessioni aggiunge una nuova connessione utente-tipo (dati utente e tipo)
    - GET /connessioni trova tutte le connessioni utente-tipo
    - GET /connessioni/{utente} trova tutte le connessioni utente-tipo di un certo utente
- Il servizio **enigmi-seguiti** consente a un utente di trovare gli enigmi di tutti i tipi che segue. Operazioni:
    - GET /enigmiseguiti/{utente} trova tutti gli enigmi seguiti da un certo utente, ovvero gli enigmi di tipi di enigmi seguiti da quell'utente (gli enigmi sono senza soluzione)
- Il servizio **api-gateway** (esposto sulla porta 8080) è l'API gateway dell'applicazione, che:
    - espone il servizio **enigmi** sul path /enigmi; ad esempio, GET /enigmi/enigmi
    - espone il servizio **connessioni** sul path /connessioni;
      ad esempio, GET /connessioni/connessioni/{utente}
    - espone il servizio **enigmi-seguiti** sul path /enigmi-seguiti;
      ad esempio, GET /enigmi-seguiti/enigmiseguiti/{utente}

Sul repository GitHub del corso, l'implementazione dell'operazione GET /enigmiseguiti/U del servizio **enigmi-seguiti**, per trovare gli enigmi seguiti dall'utente U, è basata su invocazioni remote REST ai servizi **connessioni** e **enigmi**, come segue:

- prima viene invocata l'operazione GET /connessioni/U di **connessioni** per trovare l'insieme TT dei tipi seguiti dall'utente U
- poi viene invocata l'operazione GET /cercaenigmi/tipi/TT di **enigmi**, per trovare gli enigmi dei tipi seguiti da U
- viene infine restituito l'insieme di tutti questi enigmi, che sono proprio quelli seguiti dall'utente U

Discussione

L'implementazione del servizio **enigmi-seguiti** soffre di diverse limitazioni. In particolare:

- Ogni volta che si vuole accedere agli enigmi seguiti da un certo utente è necessario movimentare in rete una grande quantità di dati.
- Inoltre, questo servizio dipende fortemente dai servizi **enigmi** e **connessioni**, e se anche uno solo di questi due servizi non è disponibile allora non lo è nemmeno il servizio **enigmi-seguiti**.

L'accesso agli enigmi seguiti da un certo utente è probabilmente l'operazione eseguita più frequentemente nel social network **Edipogram**, e pertanto va implementata in modo da sostenere prestazioni, scalabilità e disponibilità.

Progetto (attività)

Il progetto consiste nel modificare l'applicazione presente sul repository GitHub del corso, svolgendo una o più tra le seguenti attività.

Modifica del codice e della configurazione dell'applicazione

Un primo gruppo di modifiche riguarda la modifica del codice e della configurazione dell'applicazione, come segue:

- Nei servizi **enigmi** e **connessioni**, usare una base di dati MySQL o PostgreSQL al posto di HSQLDB (ciascuna da eseguire in un container Docker separato).
- Modificare la logica del servizio **enigmi-seguiti**, come descritto nel seguito.

Si potrebbe pensare di modificare la logica del servizio **enigmi-seguiti** utilizzando delle invocazioni remote *asincrone* (anziché *sincrone*), ed eseguire l'algoritmo per il calcolo degli enigmi seguiti da un utente nel modo più concorrente possibile. Anche questo approccio richiede però di movimentare una grande quantità di dati in rete quando si vogliono trovare gli enigmi seguiti da un utente. Inoltre, il servizio **enigmi-seguiti** continua a dipendere fortemente dai servizi **enigmi** e **connessioni**. Dunque, bisogna cercare una soluzione migliore.

Una soluzione probabilmente migliore consiste invece nell'invertire la logica del servizio **enigmi-seguiti**, come segue:

- Quando viene aggiunto un nuovo enigma, il servizio **enigmi** deve notificare un evento **EnigmaCreatedEvent** (con tutti i dati dell'enigma, ad eccezione della soluzione), su un canale Kafka oppure RabbitMQ (da eseguire in un container Docker).
- Quando viene aggiunta una nuova connessione utente-tipo, il servizio delle **connessioni** deve notificare un evento **ConnessioneCreatedEvent**, su un canale Kafka oppure RabbitMQ.
- Il servizio **enigmi-seguiti** deve gestire una propria base di dati (separata dalle precedenti, in un container Docker separato), con una tabella per gli enigmi e una per le connessioni utente-tipo.
- Ogni volta che il servizio **enigmi-seguiti** riceve un evento **EnigmaCreatedEvent**, allora deve aggiornare di conseguenza la propria tabella degli enigmi.
- Ogni volta che il servizio **enigmi-seguiti** riceve un evento **ConnessioneCreatedEvent**, deve aggiornare di conseguenza la relativa tabella delle connessioni.
- Il servizio **enigmi-seguiti** potrà poi rispondere alle richieste GET /enigmiseguiti/{utente} accedendo solo alla propria base di dati.

Si noti che anche questa soluzione può richiedere di movimentare molti dati in rete quando vengono aggiunti nuovi enigmi o nuove connessioni, che però sono operazioni meno frequenti che non il trovare gli enigmi seguiti da un utente. Invece l'accesso agli enigmi seguiti da un utente viene effettuato solamente nell'ambito del servizio **enigmi-seguiti**.

Una variante più complessa è la seguente:

- Il servizio **enigmi-seguiti** deve sempre gestire una propria base di dati (separata dalle precedenti, in un container Docker separato), con una tabella per gli enigmi, una per le connessioni utente-tipo, ed anche una tabella enigmiseguiti che memorizza righe (utente, idEnigma, autoreEnigma, tipoEnigma, tipoSpecificoEnigma, titoloEnigma, testoEnigma).
- Ogni volta che il servizio **enigmi-seguiti** riceve un evento **EnigmaCreatedEvent**, deve aggiornare di conseguenza la propria tabella degli enigmi e anche la tabella enigmiseguiti.
- Ogni volta che il servizio **enigmi-seguiti** riceve un evento **ConnessioneCreatedEvent**, deve aggiornare di conseguenza la relativa tabella delle connessioni e anche la tabella enigmiseguiti.
- Il servizio **enigmi-seguiti** può poi rispondere alle richieste GET /enigmiseguiti/{utente} accedendo solo alla propria tabella enigmiseguiti.

Modifica della modalità di rilascio dell'applicazione

Un secondo gruppo di modifiche riguarda la modifica della modalità di rilascio dell'applicazione, sulla base delle seguenti possibili varianti:

- Eseguire i diversi servizi in container Docker, usando Docker Compose.
- Eseguire i diversi servizi in container Docker, usando Kubernetes.
- Mandare in esecuzione più istanze di ciascun servizio.

Altri progetti

Ciascun gruppo può formulare, se vuole, una propria proposta di progetto (che deve comunque avere finalità simili a quelle dei progetti illustrati in questo documento). In ogni caso, queste proposte di progetti alternativi devono essere autorizzate preventivamente dal docente.

Modalità di svolgimento e di consegna del progetto

Il progetto va svolto in gruppi preferibilmente composti da 3-5 studenti.

Ciascun gruppo dovrà interagire con il docente in questo modo:

- Ciascun gruppo dovrà comunicare al docente, per posta elettronica, la composizione del proprio gruppo, la soluzione che si intende implementare e le tecnologie che si intendono utilizzare. Sia la soluzione che le tecnologie potrebbero essere diverse da quelle proposte in questo documento.
- Poi, ciascun gruppo dovrà realizzare e verificare (sul proprio computer) la propria applicazione distribuita.
- Infine, ciascun gruppo dovrà caricare la propria applicazione distribuita su GitHub (o altro servizio di condivisione del codice). In particolare, dovrà caricare tutto il codice sorgente dell'applicazione, oltre a ogni script necessario per la compilazione e costruzione dell'applicazione (Maven o Gradle), i file Dockerfile e gli eventuali file per Docker Compose o Kubernetes, nonché gli script per mandare in esecuzione l'applicazione.
- Al completamento del progetto, il gruppo dovrà comunicare al docente, per posta elettronica, l'URI su GitHub del codice dell'applicazione, insieme a una descrizione sintetica della soluzione effettivamente implementata, delle tecnologie effettivamente utilizzate e delle attività svolte.