Casi d’uso Payment Microservice

Di seguito vengono esposti in maniera dettagliata i tre casi d’uso più significativi dell’applicazione di gestione dei pagamenti.

Creazione di un endpoint REST (HTTP GET) per ottenere l’elenco delle transazioni registrate in uno specifico lasso di tempo, espresso in formato UNIX timestamp.

Caso d’uso UC1: Visualizza transazioni

Portata: ecommerce orientato a microservizi

Attore primario: utente amministratore (caratterizzato da X-User-ID = 0)

Parti interessate:

- Microservizio di gestione degli ordini
- Microservizio di gestione della fatturazione
- Microservizio per la gestione dei log

Pre-condizioni:

- I microservizi dell’applicazione ecomm devono essere a conoscenza dell’interfaccia REST esposta dal microservizio per la gestione dei pagamenti
- È stata stabilita per i microservizi una tipologia di comunicazione asincrona mediante l’uso di un broker Kafka (di cui sono noti topic e chiavi utilizzate)
- Esiste una collezione di pagamenti all’interno di un sistema di storage NoSQL permanente (nel nostro caso MongoDB)

Post-condizione: si vogliono restituire all’utente amministratore tutti i pagamenti registrati che sono stati effettuati nel periodo di tempo specificato tramite timestamp.

Scenario principale di successo:

1. Un http client esegue una richiesta all’endpoint */transactions* specificando come parametri della richiesta (all’interno dell’URL) i valori *fromTimestamp* ed *endTimestamp* (entrambi dovranno rispettare il formato UNIX timestamp).

2. Il microservizio di gestione dei pagamenti verifica che esista il campo “X-User-ID” all’interno dell’intestazione della richiesta http e che il valore sia impostato a 0.
3. Il microservizio verifica la correttezza dei parametri della richiesta (ovvero che siano presenti i due timestamp per specificare l’intervallo temporale dei pagamenti che si vogliono ottenere).
4. Il microservizio restituisce una http response 200 (OK) il cui body contiene l’elenco dei pagamenti in formato JSON.

Estensioni o flussi alternativi:

NOTA 1*. Qualsiasi sia lo stato del database Mongo o di Kafka, la sequenza dei controlli deve rispecchiare esattamente la sequenza espressa nello scenario di successo, ed in base a tale sequenza è possibile capire che tipologia di comportamento assumerà il microservizio sulla base della combinazione di uno o più errori della richiesta o interni al server nei confronti dell’http client e del servizio kafka.

NOTA 2*. La sequenza di controllo degli errori prevede che prima vengano verificati gli errori di tipo 40x, e solo successivamente si passa alla verifica degli errori di tipo 50x.

NOTA 3*. Se il database Mongo è attivo, ciò che viene restituito come response al client risulterà essere identico al caso specificato nello scenario principale di successo, anche se il servizio di kafka non è attivo.

2A. Il microservizio verifica l’assenza del campo “X-User-ID” all’interno dell’intestazione della richiesta http.

1. Il microservizio restituisce al client una http response con stato 401 (Unauthorized).
2. Il microservizio effettua la pubblicazione del log sul topic *logging* di kafka usando la chiave *http_errors* inserendo come corpo del messaggio lo stato 401 (oltre ai vari altri parametri richiesti come da consegna).

    a. Il microservizio non riesce ad effettuare il logging a causa di problemi con kafka (es. il servizio kafka è down):

        1. Il microservizio restituisce al client una http response con stato 401 (Unauthorized) dopo un

timeout di circa 30 secondi dovuto ai tentativi di riconnessione con il servizio kafka.

2B. Il microservizio verifica che il valore del campo “X-User-ID” non è pari a 0.

1. Il microservizio restituisce al client una http response con stato 401 (Unauthorized).
2. Il microservizio effettua la pubblicazione del log sul topic *logging* di kafka usando la chiave *http_errors* inserendo come corpo del messaggio lo stato 401 (oltre ai vari altri parametri richiesti come da consegna).

    a. Il microservizio non riesce ad effettuare il logging a causa di problemi con kafka (es. il servizio kafka è down):

        1. Il microservizio restituisce al client una http response con stato 401 (Unauthorized) dopo un timeout di circa 30 secondi dovuto ai tentativi di riconnessione con il servizio kafka.

3A. Il microservizio verifica l’impossibilità nell’estrarre i parametri dalla richiesta http.

1. Il microservizio restituisce al client una http response con stato 400 (Bad Request).
2. Il microservizio effettua la pubblicazone del log sul topic *logging* di kafka usando la chiave *http_errors* inserendo come corpo del messaggio lo stato 400 (oltre ai vari altri parametri richiesti come da consegna).

    a. Il microservizio non riesce ad effettuare il logging a causa di problemi con kafka (es. il servizio kafka è down):

        1. Il microservizio restituisce al client una http response con stato 400 (Bad Request) dopo un timeout dovuto ai tentativi di riconnessione con il servizio kafka.

4A. Il microservizio non riesce a recuperare i pagamenti dal database (es. MongoDB è down):

1. Il microservizio restituisce al client una http response con stato 200 (OK) contenente una lista vuota, dopo un timeout di circa 60 secondi dovuto ai tentativi di riconnessione con MongoDB.
2. Il microservizio effettua il logging su kafka sul topic *logging* con chiave *http_errors* contenente lo stack trace dell'errore di tipo 50x (oltre ai vari altri parametri richiesti come da consegna).

    a. Il microservizio non riesce ad effettuare il logging a causa di problemi con kafka (es. il servizio kafka è down):

        1. Il microservizio restituisce al client una http response con stato 500 (Internal Server Error) dopo un timeout di circa 30 secondi (aggiuntivi rispetto ai 60 secondi di timeout dovuti al fallimento del database) dovuto ai tentativi di riconnessione con il servizio kafka.

Creazione di un endpoint REST (HTTP POST) per la gestione delle IPN (Instant Payment Notification) generate dal sistema PayPal.

Caso d'uso UC2: IPN

Portata: ecommerce orientato a microservizi

Attore primario: sistema di IPN

Parti interessate:

- Microservizio di gestione degli ordini
- Microservizio di gestione della fatturazione
- Microservizio per la gestione dei log
- Sistema esterno per IPN (es. PayPal)

Pre-condizioni:

- I microservizi dell'applicazione ecomm devono essere a conoscenza dell'interfaccia REST esposta dal microservizio per la gestione dei pagamenti
- È stata stabilita per i microservizi una tipologia di comunicazione asincrona mediante l'uso di un broker Kafka (di cui sono noti topic e chiavi utilizzate)
- Esiste una collezione di pagamenti all'interno di un sistema di storage NoSQL permanente (nel nostro caso MongoDB)
- L'endpoint verso cui effettuare l'IPN e il microservizio rispecchiano lo standard utilizzato dal servizio esterno di IPN che si sta adottando.

Post-condizione: si vuole sviluppare un meccanismo di gestione e verifica di una IPN derivante dal servizio PayPal per convalidare un pagamento tramite un sistema di ecommerce.

Scenario principale di successo:

1. Il microservizio di gestione dei pagamenti riceve una IPN all'endpoint */ipn*.
2. Il microservizio verifica la presenza del content-type *x-www-form-urlencoded* all'interno dell'header della richiesta.
3. Il microservizio verifica la formattazione del body della richiesta, ossia che i parametri siano conformi alla specifica del content-type prima controllato.
4. Il microservizio verifica che tutti i campi necessari per la memorizzazione di un pagamento siano presenti all'interno del body della richiesta.
5. Il microservizio verifica, attraverso una richiesta REST all'endpoint di verifica esposto dal servizio PayPal (*https://ipnpb.sandbox.paypal.com/cgi-bin/webscr*), la validità della IPN (ultime due fasi dell'interazione a 4 vie).
6. Il microservizio verifica che la mail dell'account business (destinatario del pagamento) sia conforme alla mail configurata di default all'interno del microservizio stesso.
7. Il microservizio memorizza nel database il pagamento di cui è stata effettuata la IPN.

Estensioni o flussi alternativi:

NOTA 1*. Qualsiasi sia lo stato del database Mongo o di Kafka, la sequenza dei controlli deve rispecchiare esattamente la sequenza espressa nello scenario di successo, ed in base a tale sequenza è possibile capire che

tipologia di comportamento assumerà il microservizio sulla base della combinazione di uno o più errori della richiesta o interni al server nei confronti dell'http client e del servizio kafka.

NOTA 2*. Qualunque sia lo stato in cui si trova il database Mongo o il servizio kafka, la response verso il client sarà sempre una 200 OK (come da specifica del progetto) dato che questa rappresenta una risposta all'avvenuta ricezione dell'IPN.

NOTA 3*. Poiché l'operazione di IPN è gestita in maniera transazionale, le operazioni di memorizzazione nel database e di scrittura sui topic kafka avvengono se e solo se è possibile effettuarle entrambe in maniera atomica (basta che uno tra i due servizi sia down affinché l'operazione di IPN non venga effettuata, ad eccezione della restituzione di una response 200 OK al client per conferma dell'avvenuta ricezione della IPN).
L'unica eccezione si verifica nel caso in cui il DB dovesse essere down e gli errori che si verificano dovessero riguardare il content-type della richiesta, in questo caso è possibile scrivere sul topic *logging* di kafka la condizione di errore che si è verificata (di tipo 400).

2A. Il microservizio verifica l'assenza del campo *Content-Type: x-www-form-urlencoded* all'interno dell'header della richiesta, o che esso sia diverso da quanto aspettato.

1. Il microservizio restituisce al client una http response con stato 200 (OK).
2. Il microservizio effettua la pubblicazione del log sul topic *logging* di kafka usando la chiave *bad_ipn_error* inserendo come corpo del messaggio lo stato 400 (oltre ai vari altri parametri richiesti come da consegna).

    a. Il microservizio non riesce ad effettuare il logging a causa di problemi con kafka (es. il servizio kafka è down):

        1. Il microservizio restituisce al client una http response con stato 200 (OK) dopo un timeout di circa 30 secondi dovuto ai tentativi di riconnessione con il servizio kafka.

3A. Il microservizio verifica che il corpo della richiesta è incompatibile con quanto aspettato.

1. Il microservizio restituisce al client una http response con stato 200 (OK).
2. Il microservizio effettua la pubblicazione del log sul topic *logging* di kafka usando la chiave *bad_ipn_error* inserendo come corpo del messaggio lo stato 400 (oltre ai vari altri parametri richiesti come da consegna).

    a. Il microservizio non riesce ad effettuare il logging a causa di problemi con kafka (es. il servizio kafka è down):

        1. Il microservizio restituisce al client una http response con stato 200 (OK) dopo un timeout di circa 30 secondi dovuto ai tentativi di riconnessione con il servizio kafka.

4A. Il microservizio verifica l'assenza di uno o più campi necessari per la memorizzazione del pagamento.

1. Il microservizio restituisce al client una http response con stato 200 (OK).
2. Il microservizio effettua la pubblicazione del log sul topic *logging* di kafka usando la chiave *bad_ipn_error* inserendo come corpo del messaggio lo stato 400 (oltre ai vari altri parametri richiesti come da consegna).

    a. Il microservizio non riesce ad effettuare il logging a causa di problemi con kafka (es. il servizio kafka è down):

        1. Il microservizio restituisce al client una http response con stato 200 (OK) dopo un timeout di circa 30 secondi dovuto ai tentativi di riconnessione con il servizio kafka.

5A. Il microservizio verifica che la risposta alla richiesta prima effettuata contiene il testo "*INVALID*".

1. Il microservizio restituisce al client una http response con stato 200 (OK).
2. Il microservizio effettua la pubblicazione del log sul topic *logging* di kafka usando la chiave *bad_ipn_error* inserendo come corpo del messaggio lo stato 500 contenente lo stack-trace dell'errore verificatosi (oltre ai vari altri parametri richiesti come da consegna).

a. Il microservizio non riesce ad effettuare il logging a causa di problemi con kafka (es. il servizio kafka è down):

    1. Il microservizio restituisce al client una http response con stato 200 (OK) dopo un timeout di circa 30 secondi dovuto ai tentativi di riconnessione con il servizio kafka.

6A. Il microservizio verifica che la mail di business non è conforme alla mail configurata di default.

1. Il microservizio restituisce al client una http response con stato 200 (OK).
2. Il microservizio effettua la pubblicazione del log sul topic *logging* di kafka usando la chiave *received_wrong_business_paypal_payment* inserendo come corpo del messaggio lo stato 400 (oltre ai vari altri parametri richiesti come da consegna).

    a. Il microservizio non riesce ad effettuare il logging a causa di problemi con kafka (es. il servizio kafka è down):

        1. Il microservizio restituisce al client una http response con stato 200 (OK) dopo un timeout di circa 30 secondi dovuto ai tentativi di riconnessione con il servizio kafka.

Creazione di un servizio di heartbeating che verifica la disponibilità del microservizio di gestione dei pagamenti e lo stato di attività del DB.

Caso d’uso UC3: Heart-Beat Service

Portata: ecommerce orientato a microservizi

Attore primario: Microservizio di gestione dei pagamenti

Parti interessate:

- Microservizio per la detection dei fault di heart-beat

Pre-condizioni:

- Il server su cui è hostata l'applicazione del microservizio di gestione dei pagamenti (nel nostro caso Netty) deve essere multithread.
- Nel caso di deployment distribuito dell'applicazione su più microservizi, deve essere noto il microservizio (o nello specifico un suo endpoint) verso cui effettuare l'heartbeat per la notifica dello stato del microservizio di gestione dei pagamenti.

Post-condizione: si vuole sviluppare un meccanismo di heartbeating che permetta a terzi di monitorare periodicamente lo stato del microservizio (database compreso).

Scenario principale di successo:

1. Il servizio di heartbeating viene istanziato come componente della *SpringBootApplication*.
2. Il servizio si attiva non appena il contesto dell'applicazione del microservizio di gestione dei pagamenti è stato inizializzato. Esso si avvia come un thread separato.
3. Il servizio verifica che sia stato settato un *beatPeriod*.
4. Il servizio invia una Ping Request al MongoDB server.
5. Il servizio, sfruttando il paradigma di programmazione asincrona reattivo, sottoscrive l'invio dell'heartbeat alla ricezione della Ping Reply.
6. Il servizio attende un periodo pari a *beatPeriod* prima di inviare una nuova Ping Request. Si ripetono quindi i passaggi a partire dalla fase numero 3.

Estensioni o flussi alternativi:

NOTA 1*. Nel caso in cui il servizio di acknowledgment dell'heartbeat non risponda, il thread rimane in attesa di risposta ma essendo gli invii dell'heartbeat sganciati dal flusso principale di esecuzione del thread, sfruttando il paradigma reattivo di programmazione asincrona, esso continuerà indisturbato la sua attività di attesa ed invio di heartbeat.

NOTA 2*. Nel caso in cui si verifichino timeout di connessione al DB (dovuti a problemi di varia natura), il servizio di heartbeating provvederà all'invio di un beat di comunicazione della mancata attività del database (oltre alle altre informazioni inviate di default) in modo da notificare in maniera tempestiva un

eventuale inizio di malfunzionamento del database (riducendo così il periodo di comunicazione dello stato impostato tramite il *beatPeriod*).

5A. Il servizio non riceve alcuna Ping Reply da parte del MongoDB server.

1. Il servizio predispone un *hook* (gancio) all'evento di sistema generato dal fallimento dell'interazione con il DB dovuto all'impossibilità di poterlo contattare.
2. All'interno dell'*hook* ci si occupa di impostare nell'heartbeat lo stato del DB a "*down*" e lo si invia all'endpoint di verifica in maniera contestuale.