Progetto Sistemi Distribuiti 2021-2022 (traccia 2)

June 10, 2022

1 Introduzione

Questa traccia è volta alla costruzione di un'applicazione distribuita su di una rete che coinvolge ipoteticamente più di un calcolatore. Per l'implementazione del backend del progetto è possibile scegliere liberamente un linguaggio di programmazione tra:

- `C`
- `C++`
- `Common Lisp`
- `Golang`
- `Haskell`
- `Java`
- `Python`
- `Racket`
- `Rust`

Non sono accettati altri linguaggi.

Si può far uso di framework o eventuali librerie di supporto allo sviluppo. L'implementazione non è quindi legata in alcun modo ai soli strumenti che sono stati trattati durante le lezioni di laboratorio. La sperimentazione è fortemente incoraggiata a patto di non far uso di soluzioni "one click". La porzione preponderante della logica richiesta non deve essere autogenerata. È possibile svolgere il progetto individualmente o a gruppi di 2 studenti.

2 Progetto "Sistema bancario"

Lo scopo di questo progetto è quello di implementare un'applicazione che sia in grado di offrire dei servizi bancari elementari come:

- creazione di un account
- chiusura di un account
- versamento e prelievo di denaro
- spostamento di denaro da un account ad un altro, non necessariamente dello stesso proprietario

esponendone la funzionalità attraverso degli endpoint REST e pagine HTML.

2.1 Endpoint REST

Il server deve implementare i seguenti endpoint con i relativi metodi HTTP:

- `/api/account`
  - GET: restituisce la lista di tutti gli account nel sistema
  - POST: crea un nuovo account con i seguenti campi:
    - `name`
    - `surname`

    e ritorna nel body della risposta il nuovo id dell'account creato. L'id di un account è una stringa di 20 caratteri rappresentante una sequenza di bytes, generati randomicamente all'occorrenza, codificati in esadecimale (ad esempio un `accountId` potrebbe essere *1087b347f1a59277eb98*).
  - DELETE: elimina l'account con id specificato dal parametro URL `id`
- `/api/account/{accountId}`
  - GET: restituisce il nome e cognome del proprietario nonché il saldo con un elenco degli identificativi di tutte le transazioni effettuate da `accountId`, in ordine cronologico ascendente (dalla più vecchia alla più recente). Inoltre, introduce un header di risposta con chiave *X-Sistema-Bancario*. Il valore dell'header deve esprimere il nome e cognome del proprietario in formato `nome;cognome`.
  - POST: effettua un versamento di denaro con un importo specificato dalla chiave `amount` nel body della richiesta. Se `amount` è negativo, viene eseguito un prelievo. Nel caso di `amount` negativo, il server deve generare un errore se il saldo del conto non è sufficiente, informando il client dell'insuccesso. In caso di successo, nel body della risposta viene restituito il nuovo saldo del conto ed un identificativo del versamento/prelievo in formato UUID v4.
  - PUT: modifica (sovrascrive) `name` e `surname` del proprietario del conto. Nel body devono quindi essere presenti le seguenti chiavi:
    - `name`
    - `surname`
  - PATCH: modifica (sovrascrive) `name` oppure `surname` del proprietario del conto. Nel body deve quindi essere presente solamente una tra le seguenti chiavi:
    - `name`
    - `surname`
  - HEAD: restituisce nome e cognome del proprietario in un header di risposta con chiave *X-Sistema-Bancario*. Il valore dell'header deve essere in formato `nome;cognome`. Non deve essere presente alcun body di risposta.
- `/api/transfer`
  - POST: effettua uno spostamento di denaro con `amount` positivo da un account a un altro. `amount` è specificato nel body della richiesta. Il server deve generare un errore se il saldo del conto di partenza non è sufficiente, informando il client dell'insuccesso. In caso di successo, nel body della risposta vengono restituiti i nuovi saldi degli account coinvolti nella transazione distinti per `accountId` ed un identificativo della transazione in formato UUID v4. Il body della richiesta presenta quindi i seguenti campi:
    - `from`
    - `to`
    - `amount`
- `/api/divert`

  - POST: annulla una transazione con id specificato dalla chiave `id` nel body della richiesta ovvero crea una nuova transazione con un nuovo UUID v4 che inverte il trasferimento di denaro tra gli account interessati dalla transazione con identificativo `id` dell'ammontare che la transazione ha coinvolto, ma solamente se il saldo dell'account del precedente beneficiario consente questa operazione. In caso contrario genera un errore opportuno.

2.2 Frontend Web

Le risorse HTML offerte dal server devono essere offerte dagli enpoint indicati con le funzionalità seguenti:

- `/` *(l'endpoint root del server)* questa pagina HTML deve mostrare all'utente un prompt con un campo di testo in cui inserire un identificativo di un account e, alla pressione di un bottone, la pagina deve popolarsi con le informazioni riguardanti il proprietario dell'account e lo storico delle transazioni con associato eventuale destinatario dello spostamento di denaro (non presente se si trattava di un prelievo/versamento) e l'ammontare coinvolto, senza ricaricamento ovvero senza una nuova richiesta GET al solo endpoint `/`. Richieste di qualunque natura ad altri endpoint sono ovviamente concesse per recuperare le informazioni dell'account da mostrare nella pagina. La transazione più recente effettuata dall'account, e quella solamente, deve essere evidenziata in grassetto. Tutte le informazioni così come descritte devono essere visualizzate in una tabella HTML (tag `table`). In caso di inserimento di un nuovo numero di conto e successiva pressione del bottone, la pagina deve ripopolarsi con le informazioni riguardanti il nuovo account non mostrando più alcun dato del vecchio.
- `/transfer` questa pagina HTML deve mostrare all'utente un prompt con tre campi di testo in cui inserire account mittente, account destinatario e un ammontare da trasferire, e, alla pressione di un bottone, deve essere registrata nel sistema la relativa transazione. La pagina deve inoltre fornire un feedback all'utente riguardante l'esito dell'operazione

Entrambe le pagine devono avere una logica di controllo dell'input prima di inviare una richiesta al backend. Ad esempio, se un utente inserisce un account non costituito da 20 cifre esadecimali, la pagina deve fornire un errore senza tentare di inviare una richiesta al backend.

Inoltre, il backend deve sempre eseguire nuovamente il controllo dell'input prima di processare la richiesta. Se l'input non è corretto, deve restituire un errore opportuno che la pagina deve essere in grado di segnalare all'utente.

2.3 Considerazioni

Le pagine HTML e gli endpoint specificati nelle sottosezioni precedenti rappresentano un elenco necessario e sufficiente di specifiche. Ogni progetto consegnato deve implementare almeno le pagine e gli endpoint descritti. È tuttavia possibile, nonché incoraggiato, espanderlo sia con metodi HTTP aggiuntivi a quelli descritti, sia con nuove pagine ed endpoint che offrano ulteriori funzionalità pertinenti al tema del progetto. In caso di espansione del progetto oltre alle specifiche, occorre avere cura di descrivere le funzioni introdotte nel file README, come richiesto dalla sezione sottostante. A seconda del linguaggio di programmazione scelto per l'implementazione, si faccia uso di librerie opportune per la creazione e gestione degli UUID v4. Si consigliano, come esempio, le seguenti librerie:

- uuid per Golang
- Data.UUID per Haskell

Il body della richiesta può essere in formato JSON o in formato URL-encoded. Il body della risposta per un endpoint REST, se presente, deve sempre essere in formato JSON.

Tutto ciò che non è descritto nelle specifiche di implementazione è lasciato a libera interpretazione dello studente.

Per quanto concerne le pagine HTML, in fase di valutazione non viene considerato in alcun modo né lo stile della pagina, né il layout dei vari elementi HTML richiesti a patto che le funzionalità descritte siano fruibili come descritto.

La base di dati della traccia, che memorizza gli account e le transazioni, deve situarsi in memoria secondaria. È quindi possibile usare, tra le molteplici possibilità:

- un formato di serializzazione ad-hoc per il progetto
- `SQLite` o qualunque altro database embedded come ad esempio TinyDB se il progetto è implementato in Python
- uno o più file in formato `CSV`, `JSON` (multiline o meno) o anche `XML`
- un database esterno (`PostgreSQL`, `MongoDB`, `Neo4j`, `InfluxDB`, `TiDB`, etc.) a patto di fornire chiare istruzioni su come poter riprodurre l'esecuzione dell'elaborato consegnato

Progetti che fanno solamente uso di strutture dati in memoria principale non verranno considerati validi per la correzione. Almeno una parte della *fonte di verità* della traccia deve essere in memoria secondaria.

3 Modalità di consegna

La *deadline* per la consegna del progetto è fissata entro il giorno 2022-06-23 alle ore 23:59.

Per consegnare l'elaborato si usi la pagina di consegna elearning relativa alla seconda traccia. La consegna deve essere fornita tramite un file archivio di tipo **zip**: non sono ammessi altri formati di compressione. Il file deve essere chiamato aderendo al formato `MATRICOLA1_Nome1_Cognome1_MATRICOLA2_Nome2_Cognome2.zip`. Se uno studente svolge il progetto individualmente, il nome del file deve essere nel formato `MATRICOLA_Nome_Cognome.zip`. Se uno studente ha più di 1 nome o più di un cognome, indicate 1 solo nome o 1 solo cognome. Se uno studente ha dei caratteri non convenzionali nel nome o cognome, questi devono essere codificabili al più in UTF-8. Consegne che non rispettano queste regole non saranno valutate.

All'interno dell'archivio, oltre ai file ed eventuali cartelle in cui è presente l'implementazione del progetto, devono esserci i seguenti file:

1. `README`: in questo file devono essere indicati nome, cognome e matricola di ogni componente del gruppo. Eventualmente è possibile inserire una descrizione o introduzione al lavoro che il gruppo ha svolto, per meglio comprendere l'architettura del sistema implementato. In caso di espansione dalle specifiche del progetto, questo è il file dove indicare la descrizione di quanto svolto.
2. `ISTRUZIONI`: in questo file deve essere indicata una chiara modalità per poter eseguire e testare correttamente l'elaborato consegnato.

   Va segnalato, tra gli aspetti importanti, il sistema operativo e l'architettura della CPU che sono stati utilizzati per lo sviluppo, spiegando come installare eventuali dipendenze esterne del progetto.

   Le istruzioni devono essere sufficientemente esaustive per arrivare a poter eseguire l'elaborato consegnato, partendo da una macchina senza alcun software installato oltre ad un sistema operativo Linux, FreeBSD o Windows su architettura x86 a 64 bit.

   Se l'elaborato è pensato per essere eseguito su sistemi operativi diversi dai 3 citati, occorre verificare che almeno 1 dei sistemi operativi citati sia in grado di intraprenderne l'esecuzione.

È possibile scrivere i file `README` e `ISTRUZIONI` scegliendo un formato tra:

- Plain text
- Markdown
- PDF
- Org

Non sono ammessi altri formati. L'assenza di uno dei file sopralistati comporterà l'esclusione dalla correzione del progetto. Ogni consegna sarà controllata sia automaticamente sia manualmente per verificare eventuali plagi. Nel caso in cui venissero riscontrati casi di plagio, i progetti in questione saranno esclusi dalla valutazione.

4 Spunti su tecnologie interessanti

Si propone di seguito una lista di tecnologie di cui è possibile usufruire per l'implementazione del progetto. Questa lista non è esaustiva e non rappresenta un vincolo in alcun modo.

- C++ e framework drogon o Proxygen
- Golang e framework Chi oppure la sola libreria standard, in particolare i package net/http e html/template
- Scheme con implementazione Racket
- Rust con framework Rocket o con framework Actix
- Haskell con framework Scotty o con framework Yesod
- Python con framework Flask o con framework Django
- Java con framework Spring