Relazione tecnica progetto database

Anton Kozhin

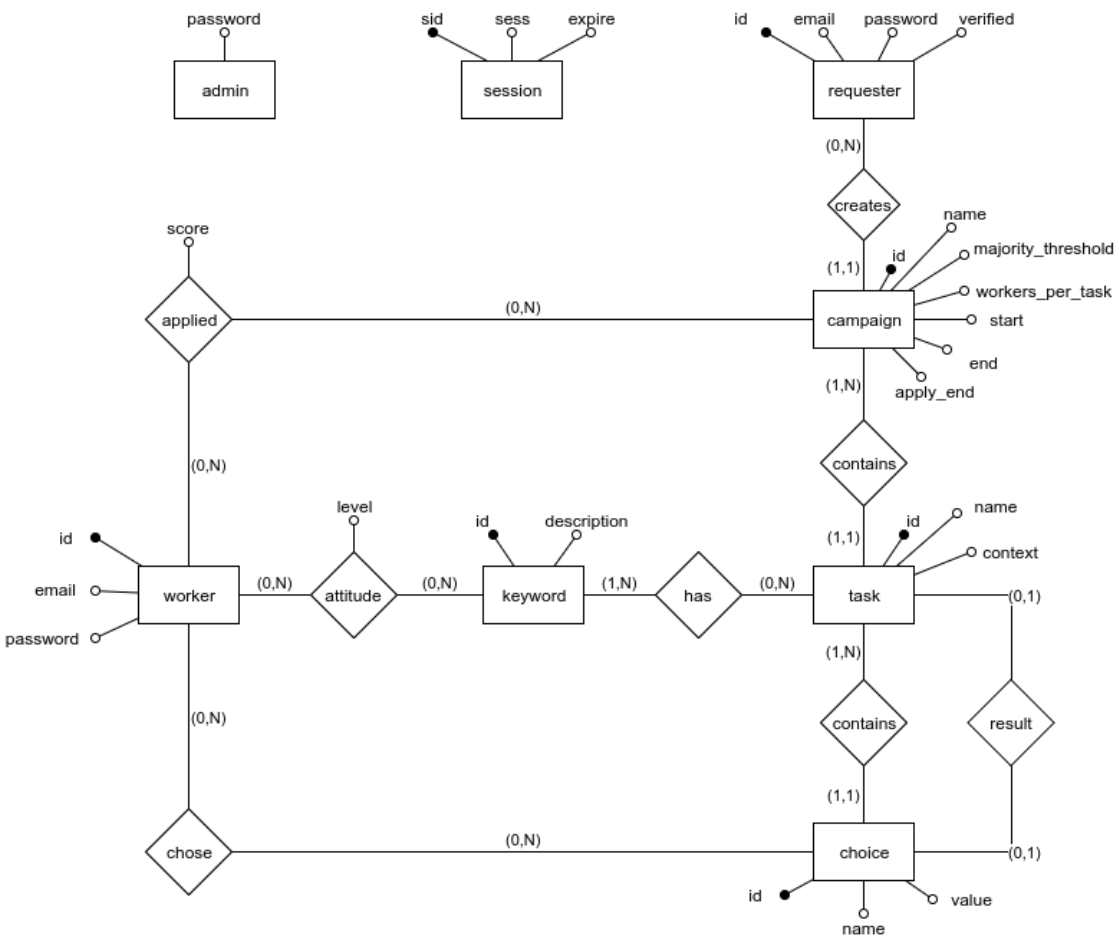
5 luglio 2018

1 Strumenti software

- Docker
- PostgreSQL
- NodeJs
- SCSS
- HTML5
- mustache
- js

2 Progettazione database

2.1 Schema ER



2.1.1 Considerazioni su schema ER

Ci sono tre tipologie di utenti, worker, requester e admin. L'admin è un utente unico con nome utente predefinito e password modificabile, mentre il numero di worker e requester non è fissato. Hanno inoltre attributi simili, ma il requester ne presenta uno in più, e hanno relazioni diverse.

Quindi ho scelto di mantenere le entità figlie invece di raggrupparle in un'entità padre. Mantenendo così le singole tabelle. Un motivo per cui ho fatto questa scelta è avere maggiore espressività dello schema della base di dati, poichè dal nome di della tabella è chiaro immediatamente a quale tipo di utente ci si riferisce. Un'altra considarazione è che le interrogazioni discriminano sempre per tipo di utente, in quanto le relazioni sono diverse tra requester e worker.

Una scelta diversa è quella di raggruppare worker, requester e admin in un'entità padre chiamata utente e discriminare a seconda dell'attributo `tipo` che può assumere i valori "worker", "requester" o "admin". Secondo me quest'ultima scelta avrebbe incrementato la complessità delle interrogazioni.

2.2 Schema relazionale

Le chiavi esterne sono in corsivo e se non specificato tra parentesi si riferiscono all'attributo `id` della tabella con lo stesso nome della chiave esterna.
Le chiavi primarie sono sottolineate.
Gli attributi opzionali sono indicati con un asterisco.

requester(id, email, password, verified)
worker(id, email, password)
campaign(id, name, majority_threshold, workers_per_task, start, end, apply_end, *requester*)
task(id, name, context, *campaign*, *result**(choice))
choice(id, name, value, *task*)
keyword(id, description)
task_keyword(*task*, *keyword*)
worker_attitude(*worker*, *keyword*, level)
worker_campaign(*worker*, *campaign*, score*)
worker_choice(worker, choice)
session(sid, sess, expire)
admin(password)

2.3 Scelte progettuali

2.3.1 Gestione keyword

Le keyword sono gestite con dei suggerimenti proposti al requester per la keyword che stà digitando. I suggerimenti sono l'insieme delle keyword tali che l'input dell'utente ne è una sottostringa. In questo modo l'utente ha tutta la liberta di scegliere le keyword che desidera tenendo sott'occhio le keyword "simili" già conosciute dall'applicazione.

Questa soluzione permette di evitare ridondanza dovuta all'uso del plurale e altri suffissi o prefissi. La scelta di suggerire keyword che contengono l'input dell'utente come sottostringa è dovuta alla disponibilita dell'operatore sql `like` e la sua semplicità di utilizzo.

2.3.2 Profilo lavoratore

Il grado di competenza/attitudine di un lavoratore rispetto ad una keyword è rappresentato da un numero intero positivo. Una keyword non associata al lavoratore si suppone abbia livello zero. Tutti i lavoratori al momento dell'iscrizione, non hanno keyword associate e quindi hanno tutte le competenze/attitudini a livello zero. Successivamente allo svolgimento di un task che risulta valido, vengono aggiornati i profili dei lavoratori che lo hanno eseguito, incrementando o decrementando di uno il livelo delle keyword associate al task.

Tutte le competenze/attitudini che si trovano a livello zero in un certo istante, vengono disassociate dal profilo del lavoratore in quanto il livello zero è sottinteso per tutte le keyword per tutti i lavoratori se non altrimenti specificato.

2.3.3 Assegnazione task

Responsabilità della funzione pl/pgsql `assign_task` nel file `db/sql/1-functions.sql`.

L'assegnazione di un task al lavoratore che ne fa richiesta all'interno di una campagna di lavoro è gestita mediante un indice detto P calcolato per ogni task della campagna di lavoro.

Dato un lavoratore che fa richiesta di un task, gli verrà assegnato il task con l'indice P massimo tra i task non completati di quella campagna.

$$P_{task} = \sum_{keyword \in task} livello\_lavoratore(keyword)$$

Cioè, l'indice P associato ad un generico task è la somma dei livelli delle keyword del task associate al lavoratore.

Per esempio un lavoratore appena iscritto ha tutte le competenze/attitudini a livello zero, quindi la scelta del task da assegnare diventa casuale. Cosicchè al lavoratore non sia preclusa la possibilità di eseguire dei task che non gli competono in parte o completamente. Al fine di permettere ai lavoratori di modellare in modo automatico il proprio profilo per rispecchiare il piu possibile la realtà.

2.3.4 Aggiornamento profilo lavoratore

Responsabilità della funzione `complete_task` nel file `db/sql/1-functions.sql`.

Questa funzione è attivata dal trigger per l'evento `insert` della tabella `worker_choice`, si occupa soprattutto di calcolare l'esito di un task e di aggiornare di conseguenza i profili dei lavoratori coinvolti qualora sia necessario.

Dato che tutti i lavoratori hanno tutte le keyword a livello zero se non altrimenti specificato. L'aggiornamento del profilo di un lavoratore avviene incrementando o decrementando di uno il livello associato alla keyword, in base all'appartenenza del lavoratore al gruppo che ha dato la risposta maggioritaria.

Se la keyword viene decrementata al livello zero viene disassociata dal lavoratore e assume il suo valore di default, altrimenti, se la keyword viene incrementata a livello uno, dovra' essere associata al lavoratore esplicitamente in quanto possiede ora un livello maggiore di zero.

2.3.5 Top 10 di una campagna di lavoro

La classifica dei dieci migliori lavoratori di una campagna è decisa in base allo score dei lavoratori nella campagna stessa.

3 Applicazione web

3.1 Connessione alla base di dati

La connessione al database avviene attraverso la libreria nodeJs *pg-promise*, che permette di effettuare interrogazioni asincrone con le *Promesse* di javaScript.

3.2 Backend

Ho utilizzato la libreria *express* per nodeJs perchè ritengo sia ideale per un'applicazione basata sulla base di dati. Principalmente perché utizza il paradigma single thread, che permette di effettuare chiamate non bloccanti alla base di dati che gestisce il carico di lavoro.

3.3 Frontend

Sviluppato principalmente in HTML5, css/scss e javaScript. Ho utilizzato la libreria css *skeleton* per aiutarmi nello stile e nell'impaginazione, e la libreria javaScript *awesomeplete* per il sistema di suggerimenti per le keyword durante la creazione di un task.

I sorgenti HTML sono nel formato di template *Mustache*, per permettere la creazione dinamica delle pagine.