Kamil Breczko

Projekt: System wspomagający organizację konferencji

13 czerwiec 2017

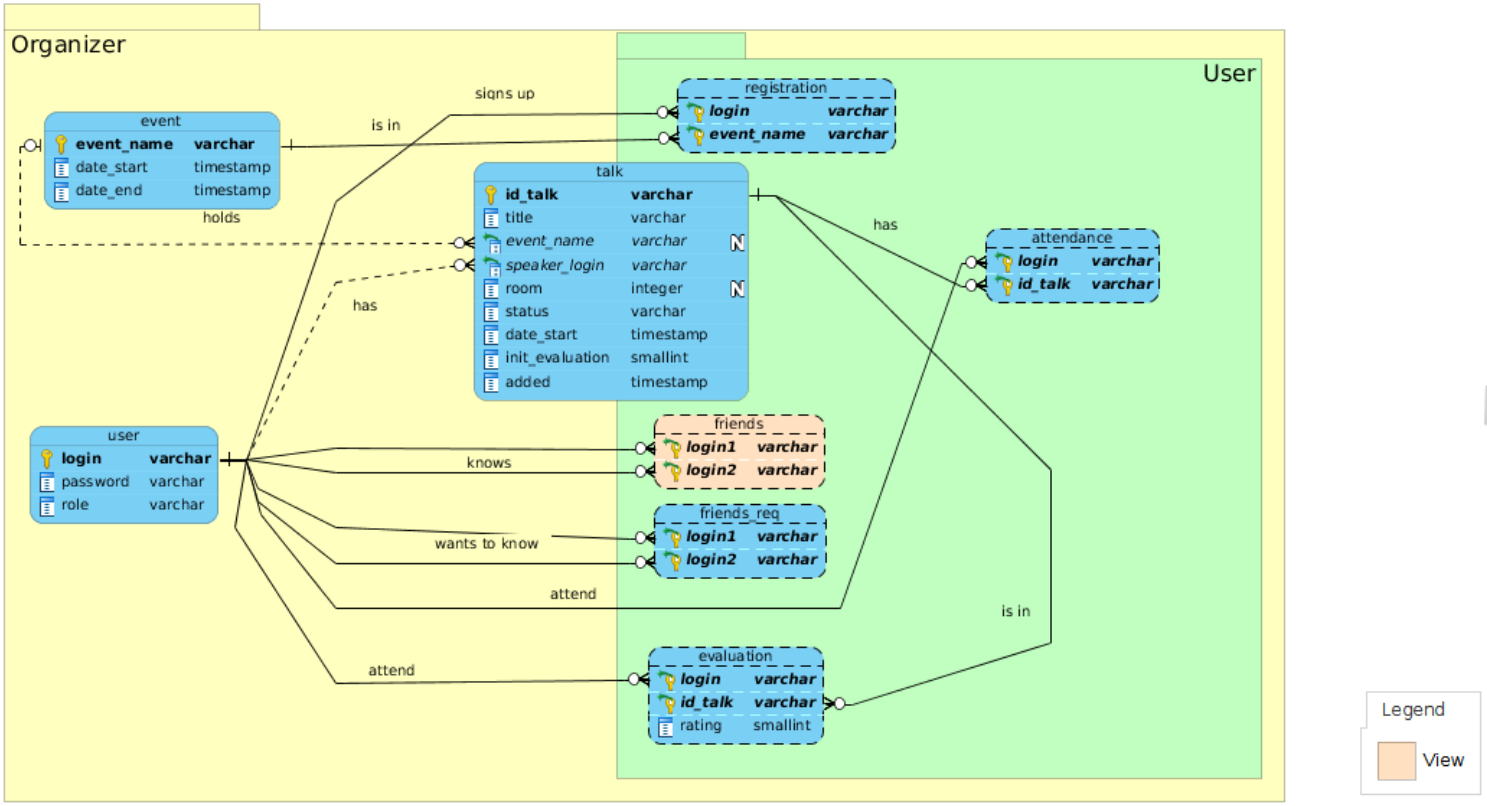
Spis treści

1. Wstęp

System wspomagający organizację konferencji. Program po uruchomieniu wczytuje ze standardowego wejścia ciąg wywołań funkcji API, a wyniki ich działania wypisać na standardowe wyjście. Jest możliwość przekazania pliku z poleceniami, które powinny się wykonać.
Wszystkie dane przechowywane są w bazie danych, efekt działania każdej funkcji modyfikującej bazę, dla której wypisano potwierdzenie wykonania (wartość OK) jest utrwalany. Program może być uruchamiany wielokrotnie np. najpierw wczytanie pliku z danymi początkowymi, a następnie kolejnych testów poprawnościowych. Przy pierwszym uruchomieniu program tworzy wszystkie niezbędne elementy bazy danych (tabele, więzy, funkcje wyzwalacze) zgodnie z przygotowanym modelem fizycznym. Zakłada się że baza danych istnieje (powinna być pusta).

2. Opis bazy danych

2.1. Diagram



2.2. Role(użytkownicy, ich schematy i akcje)

Rola uczestnika:
Uprawienia: Zapis

- rejestracja na dowolne wydarzenie;
- odnotowanie faktycznej obecności na wydarzeniu;
- wygłaszanie dowolną liczbę referatów;
- nawiązywanie znajomości z innymi uczestnikami;
- ocena referatu;
- proponowanie referatu;

Uprawienia: Odczyt

- przeglądanie referatów:
  - zaplanowanych na dany dzień;
  - znajdujących się w wydarzeniach, na których uczestnik jest zapisany;
  - najlepiej ocenianych;
  - najczęściej wybieranych(z najlepszą frekwencją) referatów;
  - wysłuchanych przez uczestnika (uczestnik był obecny);
  - odrzuconych (dla danego uczestnika);
  - wygłaszanych przez znajomych uczestnika;
  - ostatnio zatwierdzonych;
  - najbardziej polecanych;

- sprawdzenie listy znajomych uczestnika;

Rola organizatora:
Uprawienia: Zapis:
- zatwierdzenie referatu zgłoszonego przez uczestnika;
- przydzielanie sali dla danego referatu;
- publikowanie informacji o referacie;
- ocena referatu;
- usuwanie referatu spontanicznego z listy zaproponowanych;
- rejestracja wydarzeń;
- rejestracja nowego uczestnika;
- rejestracja na dowolne wydarzenie;
- odnotowanie faktycznej obecności na wydarzeniu;
- proponowanie referatu;
- nawiązywanie znajomości z innymi uczestnikami;

Uprawienia: Odczyt:
- przeglądanie referatów:
  - wysłuchanych przez organizatora (organizator był obecny);
  - wygłaszanych przez znajomych;
  - ostatnio zatwierdzonych;
  - najbardziej polecanych;
  - zaplanowanych na dany dzień;
  - znajdujących się w wydarzeniach, na których uczestnik jest zapisany;
  - najlepiej ocenianych;
  - najczęściej wybieranych(z najlepszą frekwencją);
  - mało popularnych w stosunku do liczby zapisanych uczestników na wydarzenie;
  - odrzuconych referatów oraz zaproponowanych;
- sprawdzenie listy znajomych organizatora;

2.3. Więzy

Dodatkowe więzy nie ujęte na diagramie:

Tabela **talk**
- Jeśli referat ma przydzielone wydarzenie, to data rozpoczęcia referatu musi się zawierać w wydarzeniu;
- Ocena organizatora musi się mieścić w przedziale od 0 do 10;
- Wartość w kolumnie *status* może przyjmować:
  - *public*;
  - *rejected*;
  - *waiting*;
- Migrowacja w stan *rejected* lub *public* jest możliwa tylko z stanu *waiting*, migracja z stanu *rejected* lub *public* w inny stan nie jest możliwa;
- Domyślną wartością w kolumnie *added* jest obecny czas;
- Domyślną wartością w kolumnie *status* jest *waiting*;

Tabela **user**
- Wartość w kolumnie *role* może przyjmować:
  - *organizer*;
  - *user*;
- Domyślną wartością w kolumnie *role* jest *user*

Tabela ***friends_req***
- Wartość w kolumnie *login1* musi być różna od wartości w kolumnie *login2*

Tabela **event**
- Data rozpoczęcia wydarzenia musi być "mniejsza" niż data zakończenia

Tabela **evaluation**
- Liczba w kolumnie *rating* musi być mniejsza lub równa 10 i większa lub równa 0;

3. Opis programu

3.1. Format pliku wejściowego

Każda linia pliku wejściowego zawiera obiekt JSON (http://www.json.org/json-pl.html). Każdy z obiektów opisuje wywołanie jednej funkcji API wraz z argumentami.
W pierwszej linii wejścia znajduje się wywołanie funkcji open z argumentami umożliwiającymi nawiązanie połączenia z bazą danych.

3.2. Format wyjścia

Dla każdego wywołania wypisz w osobnej linii obiekt JSON zawierający status wykonania funkcji *OK/ERROR/NOT IMPLEMENTED* oraz zwracane dane wg specyfikacji tej funkcji.

3.3. Format opisu API

Oznaczenia:
O – *wymaga autoryzacji jako organizator,*
U – *wymaga autoryzacji jako zwykły uczestnik*,
N – *nie wymaga autoryzacji,*
(*) - *wymagana na zaliczenie*

[function][arg1][arg2]. . . [argn]- *nazwa funkcji oraz nazwy jej argumentów*

3.4. Nawiązywanie połączenia i definiowanie danych organizatora

(*) open [baza] [login] [password]
przekazuje dane umożliwiające podłączenie Twojego programu do bazy - nazwę bazy, login oraz hasło, wywoływane dokładnie jeden raz, w pierwszej linii wejścia
zwraca status OK/ERROR w zależności od tego czy udało się nawiązać połączenie z bazą

(*) organizer [secret] [newlogin] [newpassword]
tworzy uczestnika [newlogin] z uprawnieniami organizatora i hasłem [newpassword], argument [secret] musi być równy d8578edf8458ce
zwraca status OK/ERROR

3.5. Operacje modyfikujące bazę

Każde z poniższych wywołań powinno zwrócić obiekt JSON zawierający wyłącznie status wykonania: *OK/ERROR/NOT IMPLEMENTED.*

(*O) event [login] [password] [eventname] [start_timestamp] [end_timestamp]
rejestracja wydarzenia, napis jest unikalny

(*O) user [login] [password] [newlogin] [newpassword]
rejestracja nowego uczestnika i służą do autoryzacji wywołującego funkcję, który musi posiadać uprawnienia organizatora, są danymi nowego uczestnika, jest unikalny

(*O) talk [login] [password] [speakerlogin] [talk] [title] [start_timestamp] [room] [initial_evaluation] [eventname]
rejestracja referatu/zatwierdzenie referatu spontanicznego, talk- jest unikalnym identyfikatorem referatu, initial_ evaluation- jest oceną organizatora w skali 0-10, ocena traktowana tak samo jak ocena uczestnika obecnego na referacie, eventname- jest nazwą wydarzenia, którego częścią jest dany referat - może być pustym napisem, co oznacza, że referat nie jest przydzielony do jakiegokolwiek wydarzenia

(*U) register_user_for_event [login] [password] [eventname]
rejestracja uczestnika [login] na wydarzenie

(*U) attendance [login] [password] [talk]
odnotowanie faktycznej obecności uczestnika [login] na referacie [talk]

(*U) evaluation [login] [password] [talk] [rating]
ocena referatu [talk] w skali 0-10 przez uczestnika [login]

(O) reject [login] [password] [talk]
usuwa referat spontaniczny z listy zaproponowanych,

(U) proposal [login] [password] [talk] [title] [start_timestamp]
propozycja referatu spontanicznego, [talk] - unikalny identyfikator referatu

(U) friends [login1] [password] [login2]
uczestnik [login1] chce nawiązać znajomość z uczestnikiem [login2], znajomość uznajemy za nawiązaną jeśli obaj uczestnicy chcą ją nawiązać tj. po wywołaniach friends [login1] [password1] [login2] i friends [login2] [password2] [login1]

3.6. Pozostałe operacje

Każde z poniższych wywołań powinno zwrócić obiekt JSON zawierający status wykonania OK/ERROR, a także tabelę data zawierającą krotki wartości atrybutów wg specyfikacji poniżej.

(*N) user_plan [login] [limit]
zwraca plan najbliższych referatów z wydarzeń, na które dany uczestnik jest zapisany (wg rejestracji na wydarzenia) posortowany wg czasu rozpoczęcia, wypisuje pierwsze referatów, przy czym 0 oznacza, że należy wypisać wszystkie
Atrybuty zwracanych krotek: [login] [talk] [start_timestamp] [title] [room]

(*N) day_plan [timestamp]
zwraca listę wszystkich referatów zaplanowanych na dany dzień posortowaną rosnąco wg sal, w drugiej kolejności wg czasu rozpoczęcia
Atrybuty zwracanych krotek:[talk] [start_timestamp] [title] [room]

(*N) best_talks [start_timestamp] [end_timestamp] [limit] [all]
zwraca referaty rozpoczynające się w danym przedziale czasowym posortowane malejąco wg średniej oceny uczestników, przy czym jeśli jest równe 1 należy wziąć pod uwagę wszystkie oceny, w przeciwnym przypadku tylko oceny uczestników, którzy byli na referacie obecni, wypisuje pierwsze referatów, przy czym 0 oznacza, że należy wypisać wszystkie.
Atrybuty zwracanych krotek:[talk] [start_timestamp] [title] [room]

(*N) most_popular_talks [start_timestamp] [end_timestamp] [limit]
zwraca referaty rozpoczynające się w podanym przedziału czasowego posortowane malejąco wg obecności, wypisuje pierwsze referatów, przy czym 0 oznacza, że należy wypisać wszystkie.
Atrybuty zwracanych krotek: [talk] [start_timestamp] [title] [room]

(*U) attended_talks [login] [password]
zwraca dla danego uczestnika referaty, na których był obecny. [talk] [start_timestamp] [title] [room]

(*O) abandoned_talks [login] [password] [limit]
zwraca listę referatów posortowaną malejąco wg liczby uczestników zarejestrowanych na wydarzenie obejmujące referat, którzy nie byli na tym referacie obecni, wypisuje pierwsze referatów, przy czym 0 oznacza, że należy wypisać wszystkie.
Atrybuty zwracanych krotek: [talk] [start_timestamp] [title] [room] [number]

(N) recently_added_talks [limit]
zwraca listę ostatnio zarejestrowanych referatów, wypisuje ostatnie referatów wg daty zarejestrowania, przy czym 0 oznacza, że należy wypisać wszystkie.
Atrybuty zwracanych krotek: [talk] [speakerlogin] [start_timestamp] [title] [room]

(U/O) rejected_talks [login] [password]
jeśli wywołujący ma uprawnienia organizatora zwraca listę wszystkich odrzuconych referatów spontanicznych, w przeciwnym przypadku listę odrzuconych referatów wywołującego ją uczestnika.
Atrybuty zwracanych krotek: [talk] [speakerlogin] [start_timestamp] [title]

(O) proposals [login] [password]
zwraca listę propozycji referatów spontanicznych do zatwierdzenia lub odrzucenia, zatwierdzenie lub odrzucenie referatu polega na wywołaniu przez organizatora funkcji talk lub reject z odpowiednimi parametrami.
Atrybuty zwracanych krotek:[talk] [speakerlogin] [start_timestamp] [title]

(U) friends_talks [login] [password] [start_timestamp] [end_timestamp] [limit]
lista referatów rozpoczynających się w podanym przedziale czasowym wygłaszanych przez znajomych danego uczestnika posortowana wg czasu rozpoczęcia, wypisuje pierwsze referatów, przy czym 0 oznacza, że należy wypisać wszystkie.
Atrybuty zwracanych krotek: [talk] [speakerlogin] [start_timestamp] [title] [room]

(U) friends_events [login] [password] [eventname]
lista znajomych uczestniczących w danym wydarzeniu.
Atrybuty zwracanych krotek:[login] [eventname] [friendlogin]

(U) recommended_talks [login] [password] [start_timestamp] [end_timestamp] [limit]
zwraca referaty rozpoczynające się w podanym przedziale czasowym, które mogą zainteresować danego uczestnika (zaproponuj parametr obliczany na podstawie dostępnych danych – ocen, obecności, znajomości itp.), wypisuje pierwsze referatów wg nalepszego , przy czym 0 oznacza, że należy wypisać wszystkie.
Atrybuty zwracanych krotek: [talk] [speakerlogin] [start_timestamp] [title] [room] [score]

4. Uruchamianie

4.1. Dane szczegółowe

Program został wykonany zgodnie z informacjami zamieszczonymi w tabeli 4.1.

Tabela 4.1. Wymagania do uruchomienia programu

Język Programowania	Java (version 1.8.0_131)
Baza Danych	psql (PostgreSQL) 9.5.6
System operacyjny	Linux (Ubuntu 16.04)

Aplikacja powinna posiadać uprawnienia do odczytu modelu fizycznego bazy danych w celu tworzenia schematów: **database.sql**.
Program korzysta z zewnętrzych bibliotek: **json.jar** (reprezentacja obiektów json) oraz **postgresql.jar** (łączenie się z bazą danych).Wszystkie pliki powinny się znajdować w folderze głównym. W folderze **src** znajdują się pliki źródłowe i wykonywalne.

4.2. Sposób uruchamiania

Aby uruchomić program, należy skompilować program za pomocą przygotowanego pliku makefile:

```
make
```

Następnie, w celu uruchomienia, wpisać w terminal:

```
make run
```

Aby przekierować polecenia z pliku do programu, należy wpisać:

```
make run -e FILE=[nazwa_pliku]
```

,gdzie [nazwa_pliku] należy wpisać poprawną ścieżkę do pliku z poleceniami (obiektami json).

Uwaga!
Program zakończy się poprawnie wtedy gdy na końcu pliku zostanie wczytany pusty wiersz.

4.3. Przykład

Na rysunku 4.3 został przedstawiony przykład poprawnego uruchomionego programu.

```
kamil@KamilPC:~/Git/Bazy$ make
javac  -classpath ./src/:json.jar:postgresql.jar src/Command.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Abandoned_talks.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Event.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Recommended_talks.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Friends_events.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Registration.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Attendance.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Friends.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Rejected_talks.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Attended_talks.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Friends_talks.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Reject.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Best_talks.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Most_popular_talks.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Talk.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Organizer.java
javac  -classpath ./src/:json.jar:postgresql.jar src/User.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Proposal.java
javac  -classpath ./src/:json.jar:postgresql.jar src/User_plan.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Day_plan.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Proposals.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Evaluation.java
javac  -classpath ./src/:json.jar:postgresql.jar src/Recently_added_talks.java
kamil@KamilPC:~/Git/Bazy$ make run -e FILE=test1.json
javac  -classpath ./src/:json.jar:postgresql.jar src/Main.java
java  -classpath ./src/:json.jar:postgresql.jar Main<test1.json
{"status":"OK"}
{"Status":"OK"}
```

Rysunek 4.3. Poprawnie uruchomiony program