namespace java com.twitter.recos.user\_tweet\_entity\_graph.thriftjava

namespace py gen.twitter.recos.user\_tweet\_entity\_graph

#@namespace scala com.twitter.recos.user\_tweet\_entity\_graph.thriftscala

#@namespace strato com.twitter.recos.user\_tweet\_entity\_graph

namespace rb UserTweetEntityGraph

include "com/twitter/recos/features/tweet.thrift"

include "com/twitter/recos/recos\_common.thrift"

enum TweetType {

Summary = 0

Photo = 1

Player = 2

Promote = 3

Regular = 4

}

enum RecommendationType {

Tweet = 0

Hashtag = 1 // Entity type

Url = 2 // Entity type

}

enum TweetEntityDisplayLocation {

MagicRecs = 0

HomeTimeline = 1

HighlightsEmailUrlRecs = 2

Highlights = 3

Email = 4

MagicRecsF1 = 5

GuideVideo = 6

MagicRecsRareTweet = 7

TopArticles = 8 // Twitter Blue most shared articles page

ContentRecommender = 9

FrigateNTab = 10

}

struct RecommendTweetEntityRequest {

// user id of the requesting user

1: required i64 requesterId

// display location from the client

2: required TweetEntityDisplayLocation displayLocation

// the recommendation entity types to return

3: required list<RecommendationType> recommendationTypes

// seed ids and weights used in left hand side

4: required map<i64,double> seedsWithWeights

// number of suggested results per recommendation entity type

5: optional map<RecommendationType, i32> maxResultsByType

// the tweet age threshold in milliseconds

6: optional i64 maxTweetAgeInMillis

// list of tweet ids to exclude from response

7: optional list<i64> excludedTweetIds

// max user social proof size per engagement type

8: optional i32 maxUserSocialProofSize

// max tweet social proof size per user

9: optional i32 maxTweetSocialProofSize

// min user social proof size per each recommendation entity type

10: optional map<RecommendationType, i32> minUserSocialProofSizes

// summary, photo, player, promote, regular

11: optional list<TweetType> tweetTypes

// the list of social proof types to return

12: optional list<recos\_common.SocialProofType> socialProofTypes

// set of groups of social proof types allowed to be combined for comparison against minUserSocialProofSizes.

// e.g. if the input is set<list<Tweet, Favorite>>, then the union of those two social proofs

// will be compared against the minUserSocialProofSize of Tweet RecommendationType.

13: optional set<list<recos\_common.SocialProofType>> socialProofTypeUnions

// the recommendations returned in the response are authored by the following users

14: optional set<i64> tweetAuthors

// the tweet engagement age threshold in milliseconds

15: optional i64 maxEngagementAgeInMillis

// the recommendations will not return any tweet authored by the following users

16: optional set<i64> excludedTweetAuthors

}

struct TweetRecommendation {

// tweet id

1: required i64 tweetId

// sum of weights of seed users who engaged with the tweet.

// If a user engaged with the same tweet twice, liked it and retweeted it, then his/her weight was counted twice.

2: required double score

// user social proofs per engagement type

3: required map<recos\_common.SocialProofType, list<i64>> socialProofByType

// user social proofs along with edge metadata per engagement type. The value of the map is a list of SocialProofs.

4: optional map<recos\_common.SocialProofType, list<recos\_common.SocialProof>> socialProofs

}

struct HashtagRecommendation {

1: required i32 id // integer hashtag id, which will be converted to hashtag string by client library.

2: required double score

// sum of weights of seed users who engaged with the hashtag.

// If a user engaged with the same hashtag twice, liked it and retweeted it, then his/her weight was counted twice.

3: required map<recos\_common.SocialProofType, map<i64, list<i64>>> socialProofByType

// user and tweet social proofs per engagement type. The key of inner map is user id, and the value of inner map is

// a list of tweet ids that the user engaged with.

}

struct UrlRecommendation {

1: required i32 id // integer url id, which will be converted to url string by client library.

2: required double score

// sum of weights of seed users who engaged with the url.

// If a user engaged with the same url twice, liked it and retweeted it, then his/her weight was counted twice.

3: required map<recos\_common.SocialProofType, map<i64, list<i64>>> socialProofByType

// user and tweet social proofs per engagement type. The key of inner map is user id, and the value of inner map is

// a list of tweet ids that the user engaged with.

}

union UserTweetEntityRecommendationUnion {

1: TweetRecommendation tweetRec

2: HashtagRecommendation hashtagRec

3: UrlRecommendation urlRec

}

struct RecommendTweetEntityResponse {

1: required list<UserTweetEntityRecommendationUnion> recommendations

}

struct SocialProofRequest {

1: required list<i64> inputTweets // Only for some tweets we need requst its social proofs.

2: required map<i64, double> seedsWithWeights // a set of seed users with weights

3: optional i64 requesterId // id of the requesting user

4: optional list<recos\_common.SocialProofType> socialProofTypes // the list of social proof types to return

}

struct SocialProofResponse {

1: required list<TweetRecommendation> socialProofResults

}

struct RecommendationSocialProofRequest {

/\*\*

\* Clients can request social proof from multiple recommendation types in a single request.

\* NOTE: Avoid mixing tweet social proof requests with entity social proof requests as the

\* underlying library call retrieves these differently.

\*/

1: required map<RecommendationType, set<i64>> recommendationIdsForSocialProof

// These will be the only valid LHS nodes used to fetch social proof.

2: required map<i64, double> seedsWithWeights

3: optional i64 requesterId

// The list of valid social proof types to return, e.g. we may only want Favorite and Tweet proofs.

4: optional list<recos\_common.SocialProofType> socialProofTypes

}

struct RecommendationSocialProofResponse {

1: required list<UserTweetEntityRecommendationUnion> socialProofResults

}

/\*\*

\* The main interface-definition for UserTweetEntityGraph.

\*/

service UserTweetEntityGraph {

RecommendTweetEntityResponse recommendTweets (RecommendTweetEntityRequest request)

/\*\*

\* Given a query user, its seed users, and a set of input tweets, return the social proofs of

\* input tweets if any.

\*

\* Currently this supports clients such as Email Recommendations, MagicRecs, and HomeTimeline.

\* In order to avoid heavy migration work, we are retaining this endpoint.

\*/

SocialProofResponse findTweetSocialProofs(SocialProofRequest request)

/\*\*

\* Find social proof for the specified RecommendationType given a set of input ids of that type.

\* Only find social proofs from the specified seed users with the specified social proof types.

\*

\* Currently this supports url social proof generation for Guide.

\*

\* This endpoint is flexible enough to support social proof generation for all recommendation

\* types, and should be used for all future clients of this service.

\*/

RecommendationSocialProofResponse findRecommendationSocialProofs(RecommendationSocialProofRequest request)

}