package com.twitter.search.common.relevance.entities;

import java.util.Optional;

import javax.annotation.Nonnull;

import com.google.common.base.Preconditions;

import org.apache.commons.lang3.builder.EqualsBuilder;

import org.apache.commons.lang3.builder.HashCodeBuilder;

import org.apache.lucene.analysis.TokenStream;

import com.twitter.search.common.util.text.TokenizerHelper;

// Represents from-user, to-user, mentions and audioSpace admins in TwitterMessage.

public final class TwitterMessageUser {

@Nonnull private final Optional<String> screenName; // a.k.a. user handle or username

@Nonnull private final Optional<String> displayName;

@Nonnull private Optional<TokenStream> tokenizedScreenName;

@Nonnull private final Optional<Long> id; // twitter ID

public static final class Builder {

@Nonnull private Optional<String> screenName = Optional.empty();

@Nonnull private Optional<String> displayName = Optional.empty();

@Nonnull private Optional<TokenStream> tokenizedScreenName = Optional.empty();

@Nonnull private Optional<Long> id = Optional.empty();

public Builder() {

}

/\*\*

\* Initialized Builder based on an existing TwitterMessageUser

\*/

public Builder(TwitterMessageUser user) {

this.screenName = user.screenName;

this.displayName = user.displayName;

this.tokenizedScreenName = user.tokenizedScreenName;

this.id = user.id;

}

/\*\*

\* Initialized Builder screen name (handle/the name following the "@") and do tokenization

\* for it.

\*/

public Builder withScreenName(Optional<String> newScreenName) {

this.screenName = newScreenName;

if (newScreenName.isPresent()) {

this.tokenizedScreenName = Optional.of(

TokenizerHelper.getNormalizedCamelcaseTokenStream(newScreenName.get()));

}

return this;

}

/\*\*

\* Initialized Builder display name

\*/

public Builder withDisplayName(Optional<String> newDisplayName) {

this.displayName = newDisplayName;

return this;

}

public Builder withId(Optional<Long> newId) {

this.id = newId;

return this;

}

public TwitterMessageUser build() {

return new TwitterMessageUser(

screenName, displayName, tokenizedScreenName, id);

}

}

/\*\* Creates a TwitterMessageUser instance with the given screen name. \*/

public static TwitterMessageUser createWithScreenName(@Nonnull String screenName) {

Preconditions.checkNotNull(screenName, "Don't set a null screen name");

return new Builder()

.withScreenName(Optional.of(screenName))

.build();

}

/\*\* Creates a TwitterMessageUser instance with the given display name. \*/

public static TwitterMessageUser createWithDisplayName(@Nonnull String displayName) {

Preconditions.checkNotNull(displayName, "Don't set a null display name");

return new Builder()

.withDisplayName(Optional.of(displayName))

.build();

}

/\*\* Creates a TwitterMessageUser instance with the given ID. \*/

public static TwitterMessageUser createWithId(long id) {

Preconditions.checkArgument(id >= 0, "Don't sent a negative user ID");

return new Builder()

.withId(Optional.of(id))

.build();

}

/\*\* Creates a TwitterMessageUser instance with the given parameters. \*/

public static TwitterMessageUser createWithNamesAndId(

@Nonnull String screenName,

@Nonnull String displayName,

long id) {

Preconditions.checkNotNull(screenName, "Use another method instead of passing null name");

Preconditions.checkNotNull(displayName, "Use another method instead of passing null name");

Preconditions.checkArgument(id >= 0, "Use another method instead of passing negative ID");

return new Builder()

.withScreenName(Optional.of(screenName))

.withDisplayName(Optional.of(displayName))

.withId(Optional.of(id))

.build();

}

/\*\* Creates a TwitterMessageUser instance with the given parameters. \*/

public static TwitterMessageUser createWithNames(

@Nonnull String screenName,

@Nonnull String displayName) {

Preconditions.checkNotNull(screenName, "Use another method instead of passing null name");

Preconditions.checkNotNull(displayName, "Use another method instead of passing null name");

return new Builder()

.withScreenName(Optional.of(screenName))

.withDisplayName(Optional.of(displayName))

.build();

}

/\*\* Creates a TwitterMessageUser instance with the given parameters. \*/

public static TwitterMessageUser createWithOptionalNamesAndId(

@Nonnull Optional<String> optScreenName,

@Nonnull Optional<String> optDisplayName,

@Nonnull Optional<Long> optId) {

Preconditions.checkNotNull(optScreenName, "Pass Optional.absent() instead of null");

Preconditions.checkNotNull(optDisplayName, "Pass Optional.absent() instead of null");

Preconditions.checkNotNull(optId, "Pass Optional.absent() instead of null");

return new Builder()

.withScreenName(optScreenName)

.withDisplayName(optDisplayName)

.withId(optId)

.build();

}

private TwitterMessageUser(

@Nonnull Optional<String> screenName,

@Nonnull Optional<String> displayName,

@Nonnull Optional<TokenStream> tokenizedScreenName,

@Nonnull Optional<Long> id) {

this.screenName = screenName;

this.displayName = displayName;

this.tokenizedScreenName = tokenizedScreenName;

this.id = id;

}

/\*\* Creates a copy of this TwitterMessageUser instance, with the given screen name. \*/

public TwitterMessageUser copyWithScreenName(@Nonnull String newScreenName) {

Preconditions.checkNotNull(newScreenName, "Don't set a null screen name");

return new Builder(this)

.withScreenName(Optional.of(newScreenName))

.build();

}

/\*\* Creates a copy of this TwitterMessageUser instance, with the given display name. \*/

public TwitterMessageUser copyWithDisplayName(@Nonnull String newDisplayName) {

Preconditions.checkNotNull(newDisplayName, "Don't set a null display name");

return new Builder(this)

.withDisplayName(Optional.of(newDisplayName))

.build();

}

/\*\* Creates a copy of this TwitterMessageUser instance, with the given ID. \*/

public TwitterMessageUser copyWithId(long newId) {

Preconditions.checkArgument(newId >= 0, "Don't set a negative user ID");

return new Builder(this)

.withId(Optional.of(newId))

.build();

}

public Optional<String> getScreenName() {

return screenName;

}

public Optional<String> getDisplayName() {

return displayName;

}

public Optional<TokenStream> getTokenizedScreenName() {

return tokenizedScreenName;

}

public Optional<Long> getId() {

return id;

}

@Override

public String toString() {

return "[" + screenName + ", " + displayName + ", " + id + "]";

}

/\*\*

\* Compares this TwitterMessageUser instance to the given object.

\*

\* @deprecated deprecated.

\*/

@Deprecated

@Override

public boolean equals(Object o) {

if (o == null) {

return false;

}

if (o == this) {

return true;

}

if (o.getClass() != getClass()) {

return false;

}

TwitterMessageUser other = (TwitterMessageUser) o;

return new EqualsBuilder()

.append(screenName, other.screenName)

.append(displayName, other.displayName)

.isEquals();

}

/\*\*

\* Returns a hash code for this TwitterMessageUser instance.

\*

\* @deprecated deprecated.

\*/

@Deprecated

@Override

public int hashCode() {

return HashCodeBuilder.reflectionHashCode(this);

}

}