package com.twitter.product\_mixer.component\_library.selector.sorter

import com.twitter.product\_mixer.core.feature.Feature

import com.twitter.product\_mixer.core.model.common.UniversalNoun

import com.twitter.product\_mixer.core.model.common.presentation.CandidateWithDetails

import scala.reflect.runtime.universe.\_

object FeatureValueSorter {

/\*\*

\* Sort by a feature value ascending. If the feature failed or is missing, use an inferred default

\* based on the type of [[FeatureValue]]. For Numeric values this is the MinValue

\* (e.g. Long.MinValue, Double.MinValue).

\*

\* @param feature feature with value to sort by

\* @param dummyImplicit due to type erasure, implicit used to disambiguate `def ascending()`

\* between def with param `feature: Feature[Candidate, FeatureValue]`

\* from def with param `feature: Feature[Candidate, Option[FeatureValue]]`

\* @param typeTag allows for inferring default value from the FeatureValue type.

\* See [[featureValueSortDefaultValue]]

\* @tparam Candidate candidate for the feature

\* @tparam FeatureValue feature value with an [[Ordering]] context bound

\*/

def ascending[Candidate <: UniversalNoun[Any], FeatureValue: Ordering](

feature: Feature[Candidate, FeatureValue]

)(

implicit dummyImplicit: DummyImplicit,

typeTag: TypeTag[FeatureValue]

): SorterProvider = {

val defaultFeatureValue: FeatureValue = featureValueSortDefaultValue(feature, Ascending)

ascending(feature, defaultFeatureValue)

}

/\*\*

\* Sort by a feature value ascending. If the feature failed or is missing, use the provided

\* default.

\*

\* @param feature feature with value to sort by

\* @param dummyImplicit due to type erasure, implicit used to disambiguate `def ascending()`

\* between def with param `feature: Feature[Candidate, FeatureValue]`

\* from def with param `feature: Feature[Candidate, Option[FeatureValue]]`

\* @tparam Candidate candidate for the feature

\* @tparam FeatureValue feature value with an [[Ordering]] context bound

\*/

def ascending[Candidate <: UniversalNoun[Any], FeatureValue: Ordering](

feature: Feature[Candidate, FeatureValue],

defaultFeatureValue: FeatureValue

)(

implicit dummyImplicit: DummyImplicit

): SorterProvider = {

val ordering = Ordering.by[CandidateWithDetails, FeatureValue](

\_.features.getOrElse(feature, defaultFeatureValue))

SorterFromOrdering(ordering, Ascending)

}

/\*\*

\* Sort by an optional feature value ascending. If the feature failed or is missing, use an

\* inferred default based on the type of [[FeatureValue]]. For Numeric values this is the MinValue

\* (e.g. Long.MinValue, Double.MinValue).

\*

\* @param feature feature with value to sort by

\* @param typeTag allows for inferring default value from the FeatureValue type.

\* See [[featureOptionalValueSortDefaultValue]]

\* @tparam Candidate candidate for the feature

\* @tparam FeatureValue feature value with an [[Ordering]] context bound

\*/

def ascending[Candidate <: UniversalNoun[Any], FeatureValue: Ordering](

feature: Feature[Candidate, Option[FeatureValue]]

)(

implicit typeTag: TypeTag[FeatureValue]

): SorterProvider = {

val defaultFeatureValue: FeatureValue = featureOptionalValueSortDefaultValue(feature, Ascending)

ascending(feature, defaultFeatureValue)

}

/\*\*

\* Sort by an optional feature value ascending. If the feature failed or is missing, use the

\* provided default.

\*

\* @param feature feature with value to sort by

\* @tparam Candidate candidate for the feature

\* @tparam FeatureValue feature value with an [[Ordering]] context bound

\*/

def ascending[Candidate <: UniversalNoun[Any], FeatureValue: Ordering](

feature: Feature[Candidate, Option[FeatureValue]],

defaultFeatureValue: FeatureValue

): SorterProvider = {

val ordering = Ordering.by[CandidateWithDetails, FeatureValue](

\_.features.getOrElse(feature, None).getOrElse(defaultFeatureValue))

SorterFromOrdering(ordering, Ascending)

}

/\*\*

\* Sort by a feature value descending. If the feature failed or is missing, use an inferred

\* default based on the type of [[FeatureValue]]. For Numeric values this is the MaxValue

\* (e.g. Long.MaxValue, Double.MaxValue).

\*

\* @param feature feature with value to sort by

\* @param dummyImplicit due to type erasure, implicit used to disambiguate `def descending()`

\* between def with param `feature: Feature[Candidate, FeatureValue]`

\* from def with param `feature: Feature[Candidate, Option[FeatureValue]]`

\* @param typeTag allows for inferring default value from the FeatureValue type.

\* See [[featureValueSortDefaultValue]]

\* @tparam Candidate candidate for the feature

\* @tparam FeatureValue feature value with an [[Ordering]] context bound

\*/

def descending[Candidate <: UniversalNoun[Any], FeatureValue: Ordering](

feature: Feature[Candidate, FeatureValue]

)(

implicit dummyImplicit: DummyImplicit,

typeTag: TypeTag[FeatureValue]

): SorterProvider = {

val defaultFeatureValue: FeatureValue = featureValueSortDefaultValue(feature, Descending)

descending(feature, defaultFeatureValue)

}

/\*\*

\* Sort by a feature value descending. If the feature failed or is missing, use the provided

\* default.

\*

\* @param feature feature with value to sort by

\* @param dummyImplicit due to type erasure, implicit used to disambiguate `def descending()`

\* between def with param `feature: Feature[Candidate, FeatureValue]`

\* from def with param `feature: Feature[Candidate, Option[FeatureValue]]`

\* @tparam Candidate candidate for the feature

\* @tparam FeatureValue feature value with an [[Ordering]] context bound

\*/

def descending[Candidate <: UniversalNoun[Any], FeatureValue: Ordering](

feature: Feature[Candidate, FeatureValue],

defaultFeatureValue: FeatureValue

)(

implicit dummyImplicit: DummyImplicit

): SorterProvider = {

val ordering = Ordering.by[CandidateWithDetails, FeatureValue](

\_.features.getOrElse(feature, defaultFeatureValue))

SorterFromOrdering(ordering, Descending)

}

/\*\*

\* Sort by an optional feature value descending. If the feature failed or is missing, use an

\* inferred default based on the type of [[FeatureValue]]. For Numeric values this is the MaxValue

\* (e.g. Long.MaxValue, Double.MaxValue).

\*

\* @param feature feature with value to sort by

\* @param typeTag allows for inferring default value from the FeatureValue type.

\* See [[featureOptionalValueSortDefaultValue]]

\* @tparam Candidate candidate for the feature

\* @tparam FeatureValue feature value with an [[Ordering]] context bound

\*/

def descending[Candidate <: UniversalNoun[Any], FeatureValue: Ordering](

feature: Feature[Candidate, Option[FeatureValue]]

)(

implicit typeTag: TypeTag[FeatureValue]

): SorterProvider = {

val defaultFeatureValue: FeatureValue =

featureOptionalValueSortDefaultValue(feature, Descending)

descending(feature, defaultFeatureValue)

}

/\*\*

\* Sort by an optional feature value descending. If the feature failed or is missing, use the

\* provided default.

\*

\* @param feature feature with value to sort by

\* @tparam Candidate candidate for the feature

\* @tparam FeatureValue feature value with an [[Ordering]] context bound

\*/

def descending[Candidate <: UniversalNoun[Any], FeatureValue: Ordering](

feature: Feature[Candidate, Option[FeatureValue]],

defaultFeatureValue: FeatureValue

): SorterProvider = {

val ordering = Ordering.by[CandidateWithDetails, FeatureValue](

\_.features.getOrElse(feature, None).getOrElse(defaultFeatureValue))

SorterFromOrdering(ordering, Descending)

}

private[sorter] def featureValueSortDefaultValue[FeatureValue: Ordering](

feature: Feature[\_, FeatureValue],

sortOrder: SortOrder

)(

implicit typeTag: TypeTag[FeatureValue]

): FeatureValue = {

val defaultValue = sortOrder match {

case Descending =>

typeOf[FeatureValue] match {

case t if t <:< typeOf[Short] => Short.MinValue

case t if t <:< typeOf[Int] => Int.MinValue

case t if t <:< typeOf[Long] => Long.MinValue

case t if t <:< typeOf[Double] => Double.MinValue

case t if t <:< typeOf[Float] => Float.MinValue

case \_ =>

throw new UnsupportedOperationException(s"Default value not supported for $feature")

}

case Ascending =>

typeOf[FeatureValue] match {

case t if t <:< typeOf[Short] => Short.MaxValue

case t if t <:< typeOf[Int] => Int.MaxValue

case t if t <:< typeOf[Long] => Long.MaxValue

case t if t <:< typeOf[Double] => Double.MaxValue

case t if t <:< typeOf[Float] => Float.MaxValue

case \_ =>

throw new UnsupportedOperationException(s"Default value not supported for $feature")

}

}

defaultValue.asInstanceOf[FeatureValue]

}

private[sorter] def featureOptionalValueSortDefaultValue[FeatureValue: Ordering](

feature: Feature[\_, Option[FeatureValue]],

sortOrder: SortOrder

)(

implicit typeTag: TypeTag[FeatureValue]

): FeatureValue = {

val defaultValue = sortOrder match {

case Descending =>

typeOf[Option[FeatureValue]] match {

case t if t <:< typeOf[Option[Short]] => Short.MinValue

case t if t <:< typeOf[Option[Int]] => Int.MinValue

case t if t <:< typeOf[Option[Long]] => Long.MinValue

case t if t <:< typeOf[Option[Double]] => Double.MinValue

case t if t <:< typeOf[Option[Float]] => Float.MinValue

case \_ =>

throw new UnsupportedOperationException(s"Default value not supported for $feature")

}

case Ascending =>

typeOf[Option[FeatureValue]] match {

case t if t <:< typeOf[Option[Short]] => Short.MaxValue

case t if t <:< typeOf[Option[Int]] => Int.MaxValue

case t if t <:< typeOf[Option[Long]] => Long.MaxValue

case t if t <:< typeOf[Option[Double]] => Double.MaxValue

case t if t <:< typeOf[Option[Float]] => Float.MaxValue

case \_ =>

throw new UnsupportedOperationException(s"Default value not supported for $feature")

}

}

defaultValue.asInstanceOf[FeatureValue]

}

}