/\*\*

\* Copyright 2016-2018 the original author or authors.

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package com.antheminc.oss.nimbus.domain.model.state.repo.db;

import static org.springframework.data.domain.ExampleMatcher.GenericPropertyMatchers.startsWith;

import java.lang.reflect.Field;

import java.util.List;

import org.apache.commons.collections.CollectionUtils;

import org.apache.commons.lang3.StringUtils;

import org.apache.commons.lang3.reflect.FieldUtils;

import org.springframework.data.domain.Example;

import org.springframework.data.domain.ExampleMatcher;

import org.springframework.data.domain.Pageable;

import org.springframework.data.mongodb.core.query.Criteria;

import org.springframework.data.mongodb.core.query.Query;

import com.antheminc.oss.nimbus.FrameworkRuntimeException;

import com.antheminc.oss.nimbus.context.BeanResolverStrategy;

import com.antheminc.oss.nimbus.domain.defn.Constants;

import com.antheminc.oss.nimbus.domain.defn.SearchNature.StartsWith;

import com.antheminc.oss.nimbus.domain.model.state.internal.AbstractListPaginatedParam.PageWrapper.PageRequestAndRespone

;

import com.antheminc.oss.nimbus.support.EnableAPIMetricCollection;

/\*\*

\* @author Rakesh Patel

\*

\*/

@EnableAPIMetricCollection

public class MongoSearchByExample extends MongoDBSearch

{

public MongoSearchByExample(BeanResolverStrategy beanResolver) {

super(beanResolver);

}

@Override

public <T> Object search(Class<T> referredClass, String alias, SearchCriteria<?> criteria) {

Query query = buildQuery(referredClass, alias, criteria.getWhere());

if(StringUtils.equalsIgnoreCase(criteria.getAggregateCriteria(),Constants.SEARCH\_REQ\_AGGREGATE\_COUNT.code)){

return getMongoOps().count(query, referredClass, alias);

}

if(criteria.getProjectCriteria() != null && StringUtils.isNotBlank(criteria.getProjectCriteria().getAlias())) {

referredClass = (Class<T>)findOutputClass(criteria, referredClass);

}

if(criteria.getPageRequest() != null) {

return findAllPageable(referredClass, alias, criteria.getPageRequest(), query);

}

return getMongoOps().find(query, referredClass, alias);

}

private <T> Query buildQuery(Class<?> referredClass, String alias, T criteria) {

if(criteria == null)

return new Query();

ExampleMatcher matcher = ExampleMatcher.matching().withIgnoreCase().withIgnoreNullValues().withIgnorePaths("version");

matcher = recurseAllFieldsAndBuildMatcher(referredClass, criteria, matcher);

Example<T> example = Example.of(criteria, matcher);

Criteria c = Criteria.byExample(example);

Query query = new Query(c);

return query;

}

private <T> PageRequestAndRespone<T> findAllPageable(Class<T> referredClass, String alias, Pageable pageRequest,

Query query) {

Query qPage = query.with(pageRequest);

List<T> results = getMongoOps().find(qPage, referredClass, alias);

if(CollectionUtils.isEmpty(results))

return null;

return new PageRequestAndRespone<T>(results, pageRequest, () -> getMongoOps().count(query, referredClass, alias));

}

// TODO - recursive matcher is not building correctly - the fieldName should be "." seperated not just the current field name.

// e.g. CMCase > Patient > firstName ==> should be built as "patientReferred.firstName", the commented code only gets the fieldName as "firstName" with recursion into patient-- need to fix

private <T> ExampleMatcher recurseAllFieldsAndBuildMatcher(Class<?> referredClass, T criteria, ExampleMatcher matcher) {

for (Field field : FieldUtils.getAllFieldsList(referredClass)) {

field.setAccessible(true);

if (field.getType().isAssignableFrom(String.class)) {

try {

String checkString = (String) FieldUtils.readField(field, criteria);

if (checkString != null && checkString.isEmpty())

matcher = matcher.withIgnorePaths(field.getName());

} catch (IllegalAccessException e) {

throw new FrameworkRuntimeException(e);

}

}

// if(!field.getType().isPrimitive() && criteria != null) {

//

try {

// recurseAllFieldsAndBuildMatcher(field.getType(), FieldUtils.readField(field, criteria), matcher);

// } catch (Exception e) {

// throw new FrameworkRuntimeException("Could not read value of field: "+field+" on object: "+criteria, e);

// }

// }

if (field.isAnnotationPresent(StartsWith.class))

matcher = matcher.withMatcher(field.getName(), startsWith());

}

return matcher;

}

}