Dokumentacion I metodave qe perdor.

1) Filter Generic me Criteria Builder → August 2021 IkubINFO

|  |
| --- |
| package *com.project.Restaurant.repository*;  import *com.project.Restaurant.utils.FieldValuePair*; import *com.project.Restaurant.utils.SearchCriteria*; import *lombok.extern.slf4j.Slf4j*; import *org.springframework.beans.factory.annotation.Autowired*; import *org.springframework.stereotype.Repository*;  import *javax.persistence.EntityManager*; import *javax.persistence.criteria.*\*; import *java.lang.reflect.Field*; import *java.util.ArrayList*; import *java.util.Arrays*; import *java.util.List*;  *@Slf4j* *@Repository* public class *Filters* {   private final *EntityManager* entityManager;   *@Autowired* public Filters(*EntityManager entityManager*) {  this.entityManager = *entityManager*;  }   public *List* genericFilter(*Class c*, *SearchCriteria searchCriteria*, *String fetch*) {   *CriteriaBuilder* cb = entityManager.getCriteriaBuilder();  *CriteriaQuery* cq = cb.createQuery(*c*);  *Root* root = cq.from(*c*);  *List*<*Predicate*> predicateList = new ArrayList<>();  *List*<*Field*> fields = *Arrays*.*asList*(*c*.getDeclaredFields());  *Fetch* f = null;   if (isStringValid(*fetch*)) {  if (isFieldPresent(fields, *fetch*)) {  f = root.fetch(*fetch*, *JoinType*.INNER);  } else {  log.info("hyn te elsi i fetchit");  *String* parentField = isObjectInside(fields, *fetch*);  if (parentField != null) {  *String* childField = *fetch*.replace(parentField, "");  log.info("Parent = {}, Child = {}", parentField, childField);  f = root.fetch(parentField).fetch(childField);  }  }  }   if (*searchCriteria* == null) {  return entityManager.createQuery(cq.select(root)).getResultList();  }   if (*searchCriteria*.getFieldValuePair() != null) {  for (*FieldValuePair* fieldValuePair : *searchCriteria*.getFieldValuePair()) {  if (isFieldPresent(fields, fieldValuePair.getField())) {  if (!isStringValid(fieldValuePair.getOperation())) {  if (fieldValuePair.getField() != null && fieldValuePair.getValue() != null) {  log.info("Field {} is present ", fieldValuePair.getField());  predicateList.add(  isBool(fieldValuePair.getValue()) ?  cb.equal(root.get(fieldValuePair.getField()), *Boolean*.*valueOf*(fieldValuePair.getValue()))  :  cb.equal(root.get(fieldValuePair.getField()), fieldValuePair.getValue())  );  }  } else {  if (!isBool(fieldValuePair.getValue())) {  if (fieldValuePair.getOperation().compareToIgnoreCase("greaterThanOrEqualTo") == 0) {  predicateList.add(cb.greaterThanOrEqualTo(root.get(fieldValuePair.getField()), fieldValuePair.getValue()));  } else if (fieldValuePair.getOperation().compareToIgnoreCase("lessThanOrEqualTo") == 0) {  predicateList.add(cb.lessThanOrEqualTo(root.get(fieldValuePair.getField()), fieldValuePair.getValue()));  }  }   }   } else {  log.info("Has to be nested");  *String* parentField = isObjectInside(fields, fieldValuePair.getField());  if (parentField != null) {  *String* childField = fieldValuePair.getField().replace(parentField, "");  log.info("ParentField name {} , childFieldName {}", parentField, childField);  if (fieldValuePair.getOperation() != null) {  if (fieldValuePair.getOperation().compareToIgnoreCase("greaterThanOrEqualTo") == 0) {  predicateList.add(  cb.greaterThanOrEqualTo(root.join(parentField).get(childField), fieldValuePair.getValue())  );  } else if (fieldValuePair.getOperation().compareToIgnoreCase("lessThanOrEqualTo") == 0) {  predicateList.add(  cb.lessThanOrEqualTo(root.join(parentField).get(childField), fieldValuePair.getValue())  );  }  } else {  log.info("Equals operation");  log.info("Parent name {}", parentField);  predicateList.add(  cb.equal(root.join(parentField).get(childField), fieldValuePair.getValue())  );  }  }  }  }  }   cq.select(root).where(predicateList.toArray(new Predicate[predicateList.size()]));   log.info("Order by {} Sort dir {}", *searchCriteria*.getOrderBy(), *searchCriteria*.getSortDirection());   if (isStringValid(*searchCriteria*.getOrderBy())) {  log.info("Order by ain't blank {} " + *searchCriteria*.getOrderBy());  if (isFieldPresent(fields, *searchCriteria*.getOrderBy())) {  log.info("Field is present {}" + *searchCriteria*.getOrderBy());  if (isStringValid(*searchCriteria*.getSortDirection())) {  log.info("Sort direction isn't blank");  cq.orderBy(  isDirAsc(*searchCriteria*.getSortDirection())  ?  cb.asc(root.get(*searchCriteria*.getOrderBy()))  :  cb.desc(root.get(*searchCriteria*.getOrderBy()))  );  } else {  cq.orderBy(  cb.desc(root.get(*searchCriteria*.getOrderBy()))  );  }  }  }  cq.distinct(true);  return entityManager.createQuery(cq).getResultList();  }   private boolean isFieldPresent(*List*<*Field*> *existing*, *String name*) {  for (*Field* field : *existing*) {  if (field.getName().compareTo(*name*) == 0) {  return true;  }  }  return false;  }   private boolean isStringValid(*String s*) {  return *s* != null && !*s*.isEmpty() && *s*.length() > 2;  }   private boolean isBool(*String s*) {  return *s*.compareToIgnoreCase("true") == 0 || *s*.compareToIgnoreCase("false") == 0;  }   */\*\**  *\* Return field name*  *\**  *\** ***@param existing*** *\** ***@param name*** *\** ***@return*** *\*/* private *String* isObjectInside(*List*<*Field*> *existing*, *String name*) {  for (*Field* field : *existing*) {  if (*name*.contains(field.getName()) && field.getName().compareToIgnoreCase("id") != 0) {  return field.getName();  }  }  return null;  }   private boolean isDirAsc(*String s*) {  return *s*.compareToIgnoreCase("asc") == 0;  } } |

Klasa Search Criteria:

|  |
| --- |
| *@Getter* *@Setter* *@NoArgsConstructor* *@ToString* public class *SearchCriteria* {  private *List*<*FieldValuePair*> fieldValuePair;  private *String* orderBy;  private *String* SortDirection;  } |

Klasa FieldValuePair:

|  |
| --- |
| *@Getter* *@Setter* *@NoArgsConstructor* public class *FieldValuePair* {  private *String* field;  private *String* value;  private *String* operation;  } |

FieldValuePair perdoret per te mbajtur ciftet key:value per te bere pjesen where ne query si dhe operacionin (veprimin qe do kryhet mbi to pra greater than a value or less than a value). Pra where name=’joan’. Pjesa Search Criteria mban liste me field value pair dhe parametrat e nevojshem per sortimin.

Metodat private ne klase kane per qellim berjen e kodit me te lexueshem me te mirembajtshem me generic dhe me te riperdorshem.

Metoda e filtrit eshte gjenerike mund te perdoret me cdo klase me cdo entitetet pavaresisht tipit te klaseses apo projektit. Mjafton te merret e te vihet keshtu sic eshte dhe te perdoret.

Fetch perdoret per te bere fetch ato qe jane lazily initialized nqs kerkohet.

Workflow:

1) Merret criteria builder nga entitiy manager (menaxhon persistance conetext)

2) Krjohet query per klasen e kaluar si argument ( Smthing.class kalohet si argument, nga kjo mund te merren fushat qe ka)

3) Krijohet rooti I querit (root in from clause)

4) Krijohet nje arrayliste qe do te mbaje nje numer te percaktuar nga perdoruesi predicates ose kushtet. Perdoret per te krijuar where clauses

5) Meren fushat e klases

6) Kontrollohet nqs fushat qe ka dhene useri gjenden te kjo klase dhe nqs po jane valide apo jo (not null not blank)

7) Nqs cdo gje ok, shtohet fusha ne predicate list ne formen enje predicate,. (Nqs eshte booleane ka nevoje per konvertim per te shmangur exceptionat ne runtime). Nqs kemi dhe fushen operation qe nuk eshte null atehere kontrollohet me cfare ngjason operacioni greater then, less then dhe me pas shtohet. Kjo pjese mund te shtohet hiqet ne kod sipas kerkeses.

8) krijohet query me predicated e deritanishem

cq.select(root).where(predicateList.toArray(new Predicate[predicateList.size()]));

e nevojshme te konvertohet ne array sepse ate tip pranon select.where

9) E njejta procedure ndiqet edhe per orderat. Check if valid check cila eshte asc apo desc dhe shtoja querit.

cq.orderBy(  
 isDirAsc(*searchCriteria*.getSortDirection())  
 ?  
 cb.asc(root.get(*searchCriteria*.getOrderBy()))  
 :  
 cb.desc(root.get(*searchCriteria*.getOrderBy()))  
);

Nqs eshte zgejdhur nje order by por asnje sort, by default zgjedhim descending sidrejtim cb.desc.Ne fund ekzkeutohet query nga entity manager dhe merret rezultari.

Suporti per querying nested objects eshte shtuar tek kjo pjese kodi:

|  |
| --- |
| else {  log.info("Has to be nested");  *String* parentField = isObjectInside(fields, fieldValuePair.getField());  if (parentField != null) {  *String* childField = fieldValuePair.getField().replace(parentField, "");  log.info("ParentField name {} , childFieldName {}", parentField, childField);  if (fieldValuePair.getOperation() != null) {  if (fieldValuePair.getOperation().compareToIgnoreCase("greaterThanOrEqualTo") == 0) {  predicateList.add(  cb.greaterThanOrEqualTo(root.join(parentField).get(childField), fieldValuePair.getValue())  );  } else if (fieldValuePair.getOperation().compareToIgnoreCase("lessThanOrEqualTo") == 0) {  predicateList.add(  cb.lessThanOrEqualTo(root.join(parentField).get(childField), fieldValuePair.getValue())  );  }  } else {  log.info("Equals operation");  log.info("Parent name {}", parentField);  predicateList.add(  cb.equal(root.join(parentField).get(childField), fieldValuePair.getValue())  );  }   }  } |

Me ndihmen e kesaj metode

|  |
| --- |
| private *String* isObjectInside(*List*<*Field*> *existing*, *String name*) {  for (*Field* field : *existing*) {  if (*name*.contains(field.getName()) && field.getName().compareToIgnoreCase("id") != 0) {  return field.getName();  }  }  return null;  } |

Kjo metode rikthen emrin e parent fieldit. Sepse tek parametri name kalohet emri I plote I fushes se marre nga perdoruesi. Kontrollohet nqs te kjo fushe gjendet si substring ndonje nga fushat e objektit tone. Nqs po, kthehet emri I fushes me qellim identifikimine fushes qe do behet join. (duhet pasur kujdes qe nqs kemi fetch type lazy fusha te jete e fetchuar me siper). Kushti per id eshte shtuar me qellim qe te mos ngaterrohet id e objektit child me ate parent. Shmang exception.

Me pas te pjesa kryesore e kodit me ndihmen e metodave te stringave vecohet emri I fushes nested.Merret operation dhe te tjera ife me rradhe.

I rendesishm qe kur kapet objekti te kapet me root.join dhe jo me root.get perndryshe jep exception.

Shtohet kushti per fetch te nje objekti brenda nested object.

|  |
| --- |
| if (isStringValid(*fetch*)) {  if (isFieldPresent(fields, *fetch*)) {  f = root.fetch(*fetch*, *JoinType*.INNER);  } else {  log.info("hyn te elsi i fetchit");  *String* parentField = isObjectInside(fields, *fetch*);  if (parentField != null) {  *String* childField = *fetch*.replace(parentField, "");  log.info("Parent = {}, Child = {}", parentField, childField);  f = root.fetch(parentField).fetch(childField);  }  }  } |

Kjo kontrollohet nga programuesi kur therret metoden. Psh kjo nevojitet ne rastin e restaurant management project kur therritet order filter. Duhet fetch ingredient nga item. Objektet jane ne rendin: Orders → item → ingredients .

Perseri ndryshimi eshte bere qe te mos varet prej emrit te fushes specifike te projektit. Mjafton qe kur te therritet metoda te jete emri I sakte I fushave psh: **itemingredient**. E nevojshme te fetch dhe parentin kur nuk eshte I tille.

**August 17 2021**- Implemented the same generic filter using Specification. Logic is identical. The only dfference is we use the JpaSpecificationExecutor’s method findAll(..) to pass the specification. The specification is created by me, and is identical with the genericFilte method. Difference is we can add pagination way too easily to it and we don’t return a .getResultList() but we return a Predicate like return cb.and(predicateArray);

in the Service layer it is being called like this:

|  |
| --- |
| public *List*<*User*> filterSpecification(*SearchCriteria searchCriteria*) {  return userRepository.findAll(new UserSpecification().userFilter(*searchCriteria*, *User*.class, "")); } |

While the actual implementation of UserSpecification is:

|  |
| --- |
| *@Slf4j* public class *UserSpecification* {  public <*T*> *Specification*<*T*> userFilter(*SearchCriteria searchCriteria*, *Class*<*T*> *c*, *String fetch*) {  return new *Specification*<*T*>() {  *@Nullable*  *@Override* public *Predicate* toPredicate(*Root*<*T*> *root*, *CriteriaQuery*<?> *cq*, *CriteriaBuilder cb*) {   *List*<*Predicate*> predicateList = new ArrayList<>();  *List*<*Field*> fields = *Arrays*.*asList*(c.getDeclaredFields());  fetch(*root*, fields, fetch);  if (searchCriteria.getFieldValuePair() == null) {  return *cb*.and(predicateList.toArray(new Predicate[0]));  }  searchCriteria.getFieldValuePair().forEach(*fieldValuePair* -> {  if (isFieldPresent(fields, *fieldValuePair*.getField())) {  operationPerform(cb, root, predicateList, *fieldValuePair*);  } else {  log.info("Has to be nested");  *String* parentField = isObjectInside(fields, *fieldValuePair*.getField());  if (parentField != null) {  nestedOperation(cb, root, predicateList, *fieldValuePair*, parentField);  }  }  });   log.info("Order by {} Sort dir {}", searchCriteria.getOrderBy(), searchCriteria.getSortDirection());  order(searchCriteria, *cb*, *cq*, *root*, fields);  *cq*.distinct(true);  return *cb*.and(predicateList.toArray(new Predicate[predicateList.size()]));  }   };  }   private <*T*> void fetch(*Root*<*T*> *root*, *List*<*Field*> *fields*, *String fetch*) {  if (isStringValid(*fetch*)) {  if (isFieldPresent(*fields*, *fetch*)) {  *root*.fetch(*fetch*, *JoinType*.INNER);  } else {  log.info("hyn te elsi i fetchit");  *String* parentField = isObjectInside(*fields*, *fetch*);  if (parentField != null) {  *String* childField = *fetch*.replace(parentField, "");  log.info("Parent = {}, Child = {}", parentField, childField);  *root*.fetch(parentField).fetch(childField);  }  }  }  }   private boolean isStringValid(*String s*) {  return *s* != null && !*s*.isEmpty() && *s*.length() > 2;  }   private boolean isFieldPresent(*List*<*Field*> *existing*, *String name*) {  for (*Field* field : *existing*) {  if (field.getName().compareTo(*name*) == 0) {  return true;  }  }  return false;  }   private *String* isObjectInside(*List*<*Field*> *existing*, *String name*) {  for (*Field* field : *existing*) {  if (*name*.contains(field.getName()) && field.getName().compareToIgnoreCase("id") != 0) {  return field.getName();  }  }  return null;  }   private boolean isDirAsc(*String s*) {  return *s*.compareToIgnoreCase("asc") == 0;  }   private <*T*> void nestedOperation(*CriteriaBuilder cb*, *Root*<*T*> *root*, *List*<*Predicate*> *predicateList*, *FieldValuePair fieldValuePair*, *String parentField*) {  *String* childField = *fieldValuePair*.getField().replace(*parentField*, "");  log.info("ParentField name {} , childFieldName {}", *parentField*, childField);  if (*fieldValuePair*.getOperation() != null) {   if (*fieldValuePair*.getOperation().compareToIgnoreCase("greaterThanOrEqualTo") == 0) {  *predicateList*.add(  *cb*.greaterThanOrEqualTo(*root*.join(*parentField*).get(childField), *fieldValuePair*.getValue())  );  } else if (*fieldValuePair*.getOperation().compareToIgnoreCase("lessThanOrEqualTo") == 0) {  *predicateList*.add(  *cb*.lessThanOrEqualTo(*root*.join(*parentField*).get(childField), *fieldValuePair*.getValue())  );  }  } else {  log.info("Equals operation");  log.info("Parent name {}", *parentField*);  *predicateList*.add(  *cb*.equal(*root*.join(*parentField*).get(childField), *fieldValuePair*.getValue())  );  }  }   private <*T*> void operationPerform(*CriteriaBuilder cb*, *Root*<*T*> *root*, *List*<*Predicate*> *predicateList*, *FieldValuePair fieldValuePair*) {  if (!isStringValid(*fieldValuePair*.getOperation())) {  if (*fieldValuePair*.getField() != null && *fieldValuePair*.getValue() != null) {  log.info("Field {} is present ", *fieldValuePair*.getField());  *predicateList*.add(  isBool(*fieldValuePair*.getValue()) ?  *cb*.equal(*root*.get(*fieldValuePair*.getField()), *Boolean*.*valueOf*(*fieldValuePair*.getValue()))  :  *cb*.equal(*root*.get(*fieldValuePair*.getField()), *fieldValuePair*.getValue())  );  }  } else {  if (!isBool(*fieldValuePair*.getValue()) && *fieldValuePair*.getOperation().compareToIgnoreCase("greaterThanOrEqualTo") == 0) {  *predicateList*.add(*cb*.greaterThanOrEqualTo(*root*.get(*fieldValuePair*.getField()), *fieldValuePair*.getValue()));  } else if (!isBool(*fieldValuePair*.getValue()) && *fieldValuePair*.getOperation().compareToIgnoreCase("lessThanOrEqualTo") == 0) {  *predicateList*.add(*cb*.lessThanOrEqualTo(*root*.get(*fieldValuePair*.getField()), *fieldValuePair*.getValue()));  }  }  }   private boolean isBool(*String s*) {  return *s*.compareToIgnoreCase("true") == 0 || *s*.compareToIgnoreCase("false") == 0;  }   private <*T*> void order(*SearchCriteria searchCriteria*, *CriteriaBuilder cb*, *CriteriaQuery*<?> *cq*, *Root*<*T*> *root*, *List*<*Field*> *fields*) {  if (isStringValid(*searchCriteria*.getOrderBy()) && isFieldPresent(*fields*, *searchCriteria*.getOrderBy())) {   if (isStringValid(*searchCriteria*.getSortDirection())) {   *cq*.orderBy(  isDirAsc(*searchCriteria*.getSortDirection())  ?  *cb*.asc(*root*.get(*searchCriteria*.getOrderBy()))  :  *cb*.desc(*root*.get(*searchCriteria*.getOrderBy()))  );  } else {  *cq*.orderBy(  *cb*.desc(*root*.get(*searchCriteria*.getOrderBy()))  );  }  }  } } |