**IFRS9**

**(International Financial Reporting Stardards)**

**Technical Specification Document**

*Version: 1.1.0*

*Last Update: Jun 29, 2021*

*Author: xuandt10*

*Update: xuandt10*

*Target Reader: Senior Engineers*

**Version History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Status | Date | Author | Comments |
| 1.1.0 | Initial Version | 29-Jun-2021 | xuandt10 | Created |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Contents

[**A.** **GENERAL INFORMATION** 5](#_Toc42006640)

[**I.** **Scope** 5](#_Toc42006641)

[**II.** **Flow Chart technical highlevel** 5](#_Toc42006642)

[**1.** **Phoenix API** 5](#_Toc42006643)

[**2.** **Intelnal API** 6](#_Toc42006643)

[**3**. **User API** 7](#_Toc42006644)

[**4**. **Schedule job API** 8](#_Toc42006645)

[**B.** **TECHNICAL SPECIFICATION** 9](#_Toc42006646)

[**I.** **GENERIC FILTER** 9](#_Toc42006647)

[1. Requirement & Purpose 9](#_Toc42006648)

[2. Flow Chart 9](#_Toc42006649)

[*2.1.* *Workflow* 9](#_Toc42006650)

[*2.2.* *Danh sách Classes trong Module* 10](#_Toc42006651)

[3. How to use 11](#_Toc42006652)

[4. Implementation 12](#_Toc42006653)

[*4.1.* *CriteriaParser* 12](#_Toc42006654)

[4.2. *GenericSpecificationsBuilder* 13](#_Toc42006655)

[4.3. *SearchOperation* 14](#_Toc42006656)

[4.4. Spec *Search* Criteria 15](#_Toc42006657)

[4.5. Generic *Specification* 17](#_Toc42006658)

[5. Test 21](#_Toc42006659)

[**I.** **USER SERVICES** 22](#_Toc42006660)

[**2.** **Requirement & Purpose** 22](#_Toc42006661)

[**3.** **Flow Chart** 22](#_Toc42006662)

[3.1. Workflow 22](#_Toc42006663)

[*3.2.* *JWT Library* 23](#_Toc42006664)

[*3.3.* *Phoenix-api tích hợp để sử dụng token như thế nào* 24](#_Toc42006665)

[*3.4.* *Thiết kế Database trong Auth cho role* 24](#_Toc42006666)

[**4.** **How to use** 25](#_Toc42006667)

[**5.** **Implementation** 25](#_Toc42006668)

[**6.** **Test** 25](#_Toc42006669)

[**II.** **SCHEDULE SERVICES** 25](#_Toc42006670)

[**1.** **Requirement & Purpose** 25](#_Toc42006671)

[**2.** **Flow Chart** 25](#_Toc42006672)

[*1.* *Workflow* 25](#_Toc42006673)

[*2.* *Thiết kế Database cho phần schedule* 27](#_Toc42006674)

[**3.** **How to use** 28](#_Toc42006675)

[**4.** **Implementation** 28](#_Toc42006676)

[**5.** **Test** 28](#_Toc42006677)

[**III.** **BASECONTROLLER + HTTP PACKAGE** 28](#_Toc42006678)

[**1.** **Requirement & Purpose** 28](#_Toc42006679)

[**2.** **Flow Chart** 28](#_Toc42006680)

[**IV.** **MULTIPLE DATABASE CONNECTION TRONG PHOENIX-API** 29](#_Toc42006681)

[**1.** **Requirement & Purpose** 29](#_Toc42006682)

[**2.** **Flow Chart** 29](#_Toc42006683)

[**V.** **SMT- CƠ CHẾ CỦA LOADERS/ENGINES VÀ BATCH/JOB** 30](#_Toc42006684)

[**1.** **Requirement & Purpose** 30](#_Toc42006685)

[**2.** **Flow Chart** 30](#_Toc42006686)

[*2.1.* *Lấy thông tin chi tiết của Batch Loader hoặc Engine Job* 30](#_Toc42006687)

[*2.2.* *Thêm mới hoặc sửa Batch/Job* 31](#_Toc42006688)

[*2.3.* *Thiết kế Database cho phần Batch/Job* 31](#_Toc42006689)

[**3.** **How to use** 32](#_Toc42006690)

[**4.** **Implementation** 32](#_Toc42006691)

[**5.** **Test** 32](#_Toc42006692)

[**VI.** **Các mục additional** 32](#_Toc42006693)

[**1.** **Requirement & Purpose** 32](#_Toc42006694)

[**2.** **Details** 32](#_Toc42006695)

[*2.1.* Caching trong Phoenix-api 32](#_Toc42006696)

[*2.2. Common Lib* 33](#_Toc42006697)

[*2.3. Bidirectional và Unidirectional* 33](#_Toc42006698)

[*2.4 . Asynchronous Thread* 33](#_Toc42006699)

[*2.5. Cấu hình message sources cho thông báo lỗi* 34](#_Toc42006700)

[*2.6. Generic api cho parameters* 34](#_Toc42006701)

[*2.7.* *Generate Code cho các generic-endpoint* 35](#_Toc42006702)

[*2.8.* *Import cho CfAssumptions* 38](#_Toc42006703)

[*2.9.* *Audit* 39](#_Toc42006704)

[*2.10.* *Manual upload file* 40](#_Toc42006705)

[**3.** **How to use** 40](#_Toc42006706)

[**4.** **Implementation** 40](#_Toc42006707)

[**5.** **Test** 40](#_Toc42006708)

[**VII.** **Data roles** 40](#_Toc42006709)

[**1.** **Requirement & Purpose** 40](#_Toc42006710)

[**2.** **Thiết kế database phần roles** 41](#_Toc42006711)

[**3.** **AOP - aspect-oriented programming** 41](#_Toc42006712)

[**VIII.** **Approval process** 43](#_Toc42006713)

[**1.** **Requirement & Purpose** 43](#_Toc42006714)

[**IX.** **Notification** 44](#_Toc42006715)

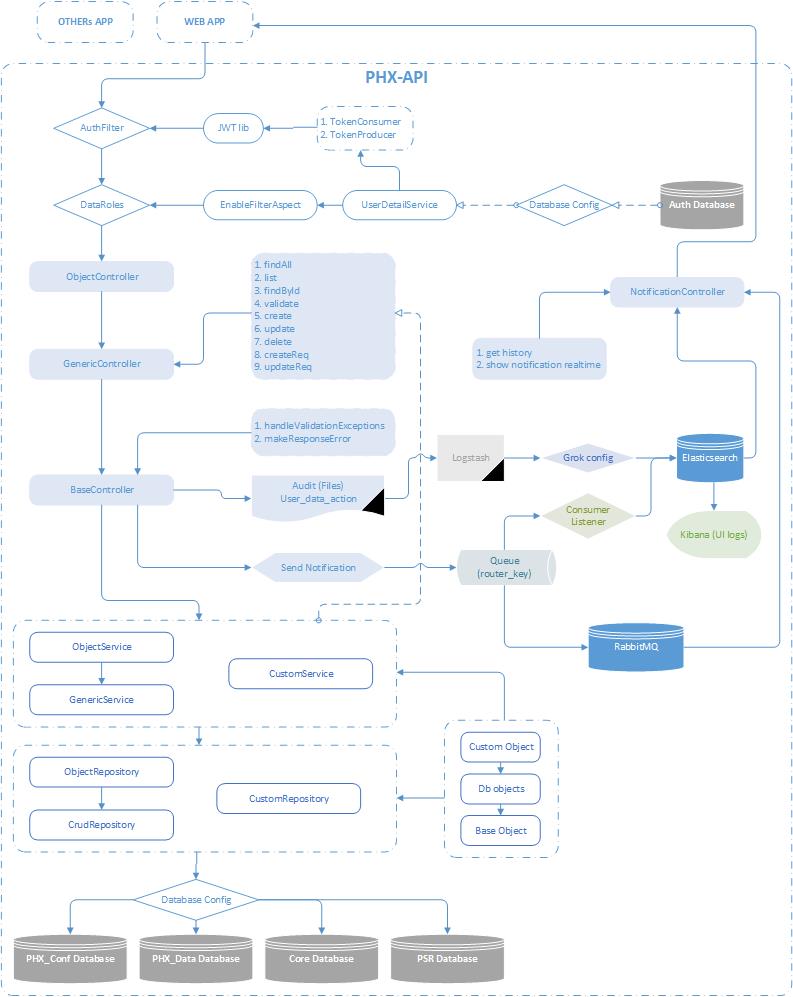
[**1.** **Flow Chart** 44](#_Toc42006716)

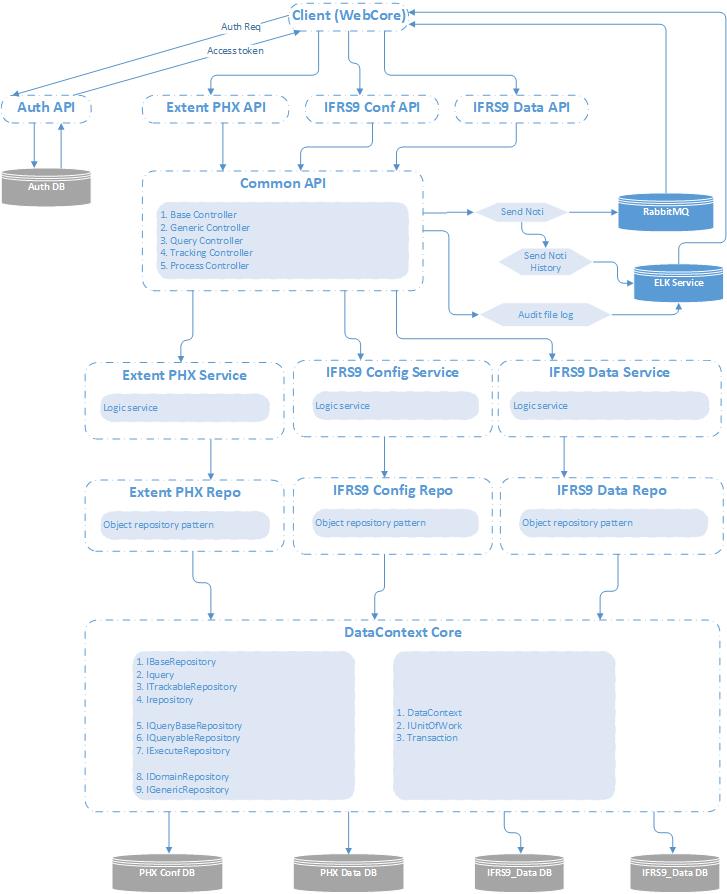
1. **GENERAL INFORMATION**
2. **Scope**

Phoenix application includes the following modules:

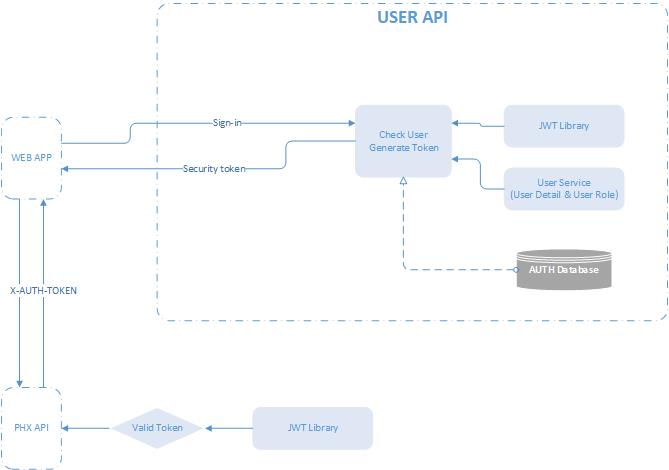
* User services
* Schedule services
* Generic filter
* Basecontroller
* Multiple database
* System management tool
* Data role
* Approval process
* Catching
* Common Lib
* Bidirectional và Unidirectional
* Asynchronous Thread
* Cấu hình message sources cho thông báo lỗi
* Generic API cho parameters
* Generate Code cho các generic-endpoint
* Import cho CfAssumptions

1. **Flow Chart technical highlevel**
2. **Phoenix API**

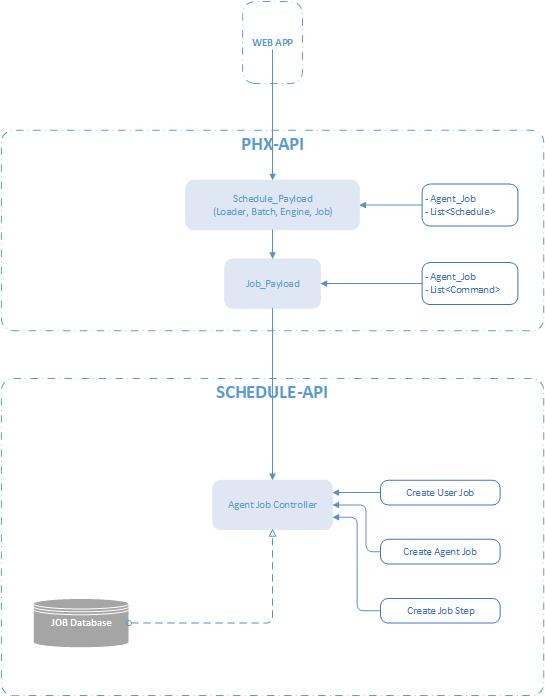


1. **.Net Core API**

## 3. User API



## 4. Schedule job API



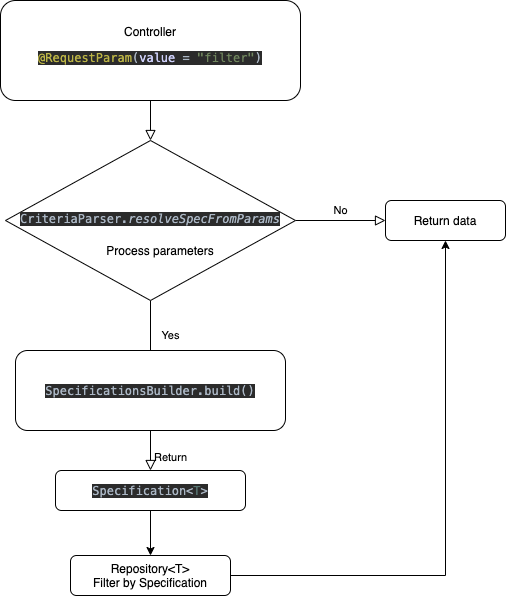
1. **TECHNICAL SPECIFICATION**
2. **GENERIC FILTER**

## Requirement & Purpose

* Hệ thống Phoenix làm việc trên các Object thay vì làm việc trực tiếp trên các Tables của SQL. Do có rất nhiều Tables cần cấu hình, giao diện sẽ cần nhiều trang có chức năng tìm kiếm và lọc điều kiện data giống nhau. Việc duplicate logic tìm kiếm và xử lý tìm kiếm cho từng pages sẽ mất nhiều thời gian maintain.
* Generic Filter Module trong Phoenix được sử dụng như một thư viện để xử lý việc tìm kiếm theo các điều kiện (equals, like, start\_with, end\_with, greater\_than, less\_than…) mà ko quan tâm đến kiểu của Object, kiểu của Column truyền vào, giống tìm kiếm của SQL và được viết dựa trên cú pháp của Apache Lucene search engine.

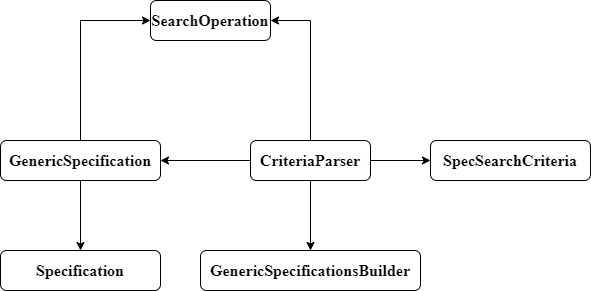
## Flow Chart

* 1. *Workflow*



* Client tích hợp với API sẽ truyền thêm *RequestParam “filter”* (require = false).
* Controller gọi hàm *CriteriaParser.resolveSpecFromParams* để trả ra kiểu Specification<T>
* Từ Repository interface cần có hàm *findAll(Specification specification)*; để lọc điều kiện theo input đưa vào. Trả kết quả cho client

* 1. *Danh sách Classes trong Module*



* **CriteriaParser**: parse điều kiện (equals, like, start\_with, end\_with, greater\_than, less\_than…), trả ra Stack. Nhóm các điều kiện “AND”, “OR” và xử lý thứ tự của các dấu ngoặc “(“, “)” trong điều kiện. Các method:
  + Deque<?> parse(String searchParam);
  + <T> Specification<T> resolveSpecFromParams(String searchParameters)
* **GenericSpecification**: override điều kiện của Specification<T>. Specification là interface của Spring Data, chỉ hỗ trợ giới hạn với từng Object cụ thể. Đây là nơi để xử lý logic filter generic, xử lý các kiểu data types đặc biệt như Boolean, Date, xử lý việc tìm kiếm theo cột trên 1 Object dùng lệnh Join. Các method:
* @Override Predicate toPredicate();
* Predicate processFilterFieldBelongOtherTable();
* checkFieldBelongOtherTable();
* Predicate getPredicate();
* convertJsonPropToEntityField();
* Date convertValueToDate(Object valueSearch)
* Object convertValueToBoolean()
* **GenericSpecificationsBuilder**: Nhận đầu vào là Stack của bộ điều kiện filter truyền vào. Trả ra kiểu Specification<T>. Các method:
* Specification<T> build();
* **SearchOperation**: Định nghĩa các kiểu tìm kiếm. Hiện tại danh sách gồm:
* EQUALITY,
* NEGATION,
* GREATER\_THAN,
* LESS\_THAN,
* LIKE,
* STARTS\_WITH,
* ENDS\_WITH,
* CONTAINS,
* NOT\_LIKE,
* NOT\_STARTS\_WITH,
* NOT\_ENDS\_WITH,
* NOT\_CONTAINS,
* GREATER\_THAN\_EQUALS,
* LESS\_THAN\_EQUALS;
* **SpecSearchCriteria**: Model chứa các điều kiện đã được parse ở bước trên.

## How to use

*{URL}:8096/engines?filter=id!NULL*

*{URL}:8096/engines?filter=id:NULL*

*{URL}:8096/engines?filter=id>20*

*{URL}:8096/engines?filter=id<20*

*{URL}:8096/engines?filter=id!20*

*{URL}:8096/engines?filter=engineName:"SIRR\_SBV\_VPBANK"*

*{URL}:8096/engines?filter=engineName!"SIRR\_SBV\_VPBANK"*

*{URL}:8096/engines?filter=isActive:false*

*{URL}:8096/engines?filter=isActive:true*

*{URL}:8096/engines?filter=fkSysEngine:\*"CF"*

*{URL}:8096/engines?filter=fkSysEngine:\*"SIRR"\**

*{URL}:8096/engines?filter=fkSysEngine:"L"\**

*{URL}:8096/engines?filter=id>24 OR id:1*

*{URL}:8096/engines?filter=id>24 OR ( id:1 AND maxExecTime>500 )*

*{URL}:8096/engineLogs?filter=businessDate:"2019-03-19"*

*{URL}:8096/engineLogs?filter=businessDate>"2019-03-19"*

*{URL}:8096/engineLogs?filter=businessDate<"2019-03-19"*

*{URL}:8096/engineLogs?filter=engineDesc:\*"LCCF CE"\* AND sessionId:5 OR ( businessDate>"2018-11-15" AND id:401 )*

*{URL}.vpbank.com.vn:8096/engines?page=0&size=1&filter=id>22 Or id<6*

## Implementation

### *CriteriaParser*

|  |
| --- |
| public class CriteriaParser {  private static Map<String, Operator> *ops*;  private static Pattern *SpecCriteraRegex* = Pattern.*compile*("^(\\w+?)(" + Joiner.*on*("|")  .join(SearchOperation.*SIMPLE\_OPERATION\_SET*) + ")(\\p{Punct}?)(.+?)(\\p{Punct}?)$", Pattern.*UNICODE\_CHARACTER\_CLASS*);  private enum Operator {  *OR*(1), *AND*(2);  final int precedence;  Operator(int p) {  precedence = p;  }  }  static {  Map<String, Operator> tempMap = new HashMap<>();  tempMap.put("AND", Operator.*AND*);  tempMap.put("OR", Operator.*OR*);  tempMap.put("or", Operator.*OR*);  tempMap.put("and", Operator.*AND*);  tempMap.put("Or", Operator.*OR*);  tempMap.put("And", Operator.*AND*);  *ops* = Collections.*unmodifiableMap*(tempMap);  }  private static boolean isHigherPrecedenceOperator(String currOp, String prevOp) {  return (*ops*.containsKey(prevOp) && *ops*.get(prevOp).precedence >= *ops*.get(currOp).precedence);  }  public static Deque<?> parse(String searchParam) {  Deque<Object> output = new LinkedList<>();  Deque<String> stack = new LinkedList<>();  for (String token : searchParam.split("\\s+(?=((\\\\[\\\\\"]|[^\\\\\"])\*\"(\\\\[\\\\\"]|[^\\\\\"])\*\")\*(\\\\[\\\\\"]|[^\\\\\"])\*$)")) {  if (*ops*.containsKey(token)) {  while (!stack.isEmpty() && *isHigherPrecedenceOperator*(token, stack.peek()))  output.push(stack.pop().equalsIgnoreCase(SearchOperation.*OR\_OPERATOR*) ? SearchOperation.*OR\_OPERATOR* : SearchOperation.*AND\_OPERATOR*);  stack.push(token.equalsIgnoreCase(SearchOperation.*OR\_OPERATOR*) ? SearchOperation.*OR\_OPERATOR* : SearchOperation.*AND\_OPERATOR*);  } else if (token.equals(SearchOperation.*LEFT\_PARANTHESIS*)) {  stack.push(SearchOperation.*LEFT\_PARANTHESIS*);  } else if (token.equals(SearchOperation.*RIGHT\_PARANTHESIS*)) {  while (!stack.peek().equals(SearchOperation.*LEFT\_PARANTHESIS*))  output.push(stack.pop());  stack.pop();  } else {  Matcher matcher = *SpecCriteraRegex*.matcher(token);  while (matcher.find()) {  output.push(new SpecSearchCriteria(matcher.group(1), matcher.group(2), matcher.group(3), matcher.group(4), matcher.group(5)));  }  }  }  while (!stack.isEmpty())  output.push(stack.pop());  return output;  }  public static <T> Specification<T> resolveSpecFromParams(String searchParameters) {  if(searchParameters == null) return null;  GenericSpecificationsBuilder<T> specificationsBuilder = new GenericSpecificationsBuilder<>();  return specificationsBuilder.build(*parse*(searchParameters), GenericSpecification::new);  }  } |

### *GenericSpecificationsBuilder*

|  |
| --- |
| public class GenericSpecificationsBuilder<T> {  public Specification<T> build(Deque<?> postFixedExprStack, Function<SpecSearchCriteria, Specification<T>> converter) {  Deque<Specification<T>> specStack = new LinkedList<>();  Collections.*reverse*((List<?>) postFixedExprStack);  while (!postFixedExprStack.isEmpty()) {  Object mayBeOperand = postFixedExprStack.pop();  if (!(mayBeOperand instanceof String)) {  specStack.push(converter.apply((SpecSearchCriteria) mayBeOperand));  } else {  Specification<T> operand1 = specStack.pop();  Specification<T> operand2 = specStack.pop();  if (mayBeOperand.equals(SearchOperation.*AND\_OPERATOR*))  specStack.push(Specification.*where*(operand1).and(operand2));  else if (mayBeOperand.equals(SearchOperation.*OR\_OPERATOR*))  specStack.push(Specification.*where*(operand1).or(operand2));  }  }  return specStack.pop();  }  } |

### *SearchOperation*

|  |
| --- |
| public enum SearchOperation {  *EQUALITY*, *NEGATION*, *GREATER\_THAN*, *LESS\_THAN*,  *LIKE*, *STARTS\_WITH*, *ENDS\_WITH*, *CONTAINS*,  *NOT\_LIKE*, *NOT\_STARTS\_WITH*, *NOT\_ENDS\_WITH*, *NOT\_CONTAINS*,  *GREATER\_THAN\_EQUALS*, *LESS\_THAN\_EQUALS*;  public static final String[] *SIMPLE\_OPERATION\_SET* = { ":", "!", ">", "<", "~", ">:", "<:"};  public static final String *OR\_PREDICATE\_FLAG* = "'";  public static final String *ZERO\_OR\_MORE\_REGEX* = "\*";  public static final String *COLON* = ":";  public static final String *OR\_OPERATOR* = "OR";  public static final String *AND\_OPERATOR* = "AND";  public static final String *LEFT\_PARANTHESIS* = "(";  public static final String *RIGHT\_PARANTHESIS* = ")";  public static SearchOperation getSimpleOperation(final String input) {  switch (input) {  case ":":  return *EQUALITY*;  case "!":  return *NEGATION*;  case ">":  return *GREATER\_THAN*;  case "<":  return *LESS\_THAN*;  case "~":  return *LIKE*;  default:  return null;  }  }  } |

### Spec *Search* Criteria

|  |
| --- |
| public class SpecSearchCriteria {  private String key;  private SearchOperation operation;  private Object value;  private boolean orPredicate;  public SpecSearchCriteria() {  }  public SpecSearchCriteria(final String key, final SearchOperation operation, final Object value) {  super();  this.key = key;  this.operation = operation;  this.value = value;  }  public SpecSearchCriteria(final String orPredicate, final String key, final SearchOperation operation, final Object value) {  super();  this.orPredicate = orPredicate != null && orPredicate.equals(SearchOperation.*OR\_PREDICATE\_FLAG*);  this.key = key;  this.operation = operation;  this.value = value;  }  public SpecSearchCriteria(String key, String operation, String prefix, String value, String suffix) {  SearchOperation op = SearchOperation.*getSimpleOperation*(operation);  value = value.replaceFirst("\"", "").replaceFirst("(?s)\"(?!.\*?\")", "");  if (op != null) {  if (op == SearchOperation.*EQUALITY*) {  final boolean startWithAsterisk = prefix != null && prefix.contains(SearchOperation.*ZERO\_OR\_MORE\_REGEX*);  final boolean endWithAsterisk = suffix != null && suffix.contains(SearchOperation.*ZERO\_OR\_MORE\_REGEX*);  if (startWithAsterisk && endWithAsterisk) {  op = SearchOperation.*CONTAINS*;  } else if (startWithAsterisk) {  op = SearchOperation.*ENDS\_WITH*;  } else if (endWithAsterisk) {  op = SearchOperation.*STARTS\_WITH*;  }  }  if (op == SearchOperation.*NEGATION*) {  final boolean startWithAsterisk = prefix != null && prefix.contains(SearchOperation.*ZERO\_OR\_MORE\_REGEX*);  final boolean endWithAsterisk = suffix != null && suffix.contains(SearchOperation.*ZERO\_OR\_MORE\_REGEX*);  if (startWithAsterisk && endWithAsterisk) {  op = SearchOperation.*NOT\_CONTAINS*;  } else if (startWithAsterisk) {  op = SearchOperation.*NOT\_ENDS\_WITH*;  } else if (endWithAsterisk) {  op = SearchOperation.*NOT\_STARTS\_WITH*;  }  }  if(op == SearchOperation.*GREATER\_THAN* || op == SearchOperation.*LESS\_THAN*) {  final boolean startWithColon = prefix != null && prefix.contains(SearchOperation.*COLON*);  if(startWithColon && op == SearchOperation.*GREATER\_THAN*) {  op = SearchOperation.*GREATER\_THAN\_EQUALS*;  }  if(startWithColon && op == SearchOperation.*LESS\_THAN*) {  op = SearchOperation.*LESS\_THAN\_EQUALS*;  }  }  }  this.key = key;  this.operation = op;  this.value = value;  }  public String getKey() {  return key;  }  public void setKey(final String key) {  this.key = key;  }  public SearchOperation getOperation() {  return operation;  }  public void setOperation(final SearchOperation operation) {  this.operation = operation;  }  public Object getValue() {  return value;  }  public void setValue(final Object value) {  this.value = value;  }  public boolean isOrPredicate() {  return orPredicate;  }  public void setOrPredicate(boolean orPredicate) {  this.orPredicate = orPredicate;  }  } |

### Generic *Specification*

|  |
| --- |
| public class GenericSpecification<T> implements Specification<T> {  private static Logger *LOGGER* = LoggerFactory.*getLogger*(GenericSpecification.class);  private SpecSearchCriteria criteria;  public GenericSpecification(final SpecSearchCriteria criteria) {  super();  this.criteria = criteria;  }  public SpecSearchCriteria getCriteria() {  return criteria;  }  @Override  public Predicate toPredicate(final Root<T> root, final CriteriaQuery<?> query, final CriteriaBuilder builder) {  Object valueSearch = null;  Date requestDate = null;  try {  convertJsonPropToEntityField(root);  valueSearch = convertValueToBoolean();  requestDate = convertValueToDate(valueSearch);  checkFieldBelongOtherTable(root);  return getPredicate(builder, valueSearch, requestDate, root);  } catch (IllegalArgumentException ex) {  return processFilterFieldBelongOtherTable(root, builder, valueSearch, requestDate, ex);  } catch (Exception ex) {  *LOGGER*.error("Build Predicate Exception", ex);  throw new BusinessException(*INCORRECT\_FILTER\_PARAMS*);  }  }  private Predicate processFilterFieldBelongOtherTable(Root<T> root, CriteriaBuilder builder, Object valueSearch, Date requestDate, IllegalArgumentException ex) {  String cap = StringUtil.*upperCaseFirstLetter*(criteria.getKey());  BelongObject annotation;  try {  Method method = root.getJavaType().getDeclaredMethod(String.*format*("get%s", cap));  annotation = method.getAnnotation(BelongObject.class);  if(annotation == null) {  *LOGGER*.error("No Annotation @BelongObject Marked", ex);  throw new BusinessException(*SERVICE\_UNVAILABLE*);  }  } catch (NoSuchMethodException e) {  *LOGGER*.error("Build Predicate Exception", ex);  throw new BusinessException(*INCORRECT\_FILTER\_PARAMS*);  }  Join<T, Object> join = root.join(annotation.value());  return getPredicate(builder, valueSearch, requestDate, join);  }  private void checkFieldBelongOtherTable(Root<T> root) {  root.get(criteria.getKey());  }  private Predicate getPredicate(CriteriaBuilder builder, Object valueSearch, Date requestDate, Root<T> root) {  switch (criteria.getOperation()) {  case *EQUALITY*:  if ("null".equalsIgnoreCase(String.*valueOf*(valueSearch)))  return builder.isNull(root.get(criteria.getKey()));  if (isMatchesDateIsoFormat(valueSearch))  return builder.equal(root.get(criteria.getKey()), requestDate);  return builder.equal(root.get(criteria.getKey()), valueSearch);  case *NEGATION*:  if ("null".equalsIgnoreCase(String.*valueOf*(valueSearch)))  return builder.isNotNull(root.get(criteria.getKey()));  if (isMatchesDateIsoFormat(valueSearch))  return builder.or(builder.notEqual(root.get(criteria.getKey()), requestDate), builder.isNull(root.get(criteria.getKey())));  return builder.or(builder.notEqual(root.get(criteria.getKey()), valueSearch), builder.isNull(root.get(criteria.getKey())));  case *GREATER\_THAN*:  if (isMatchesDateIsoFormat(valueSearch))  return builder.greaterThan(root.get(criteria.getKey()), requestDate);  return builder.greaterThan(root.get(criteria.getKey()), String.*valueOf*(valueSearch));  case *GREATER\_THAN\_EQUALS*:  if (isMatchesDateIsoFormat(valueSearch))  return builder.greaterThanOrEqualTo(root.get(criteria.getKey()), requestDate);  return builder.greaterThanOrEqualTo(root.get(criteria.getKey()), String.*valueOf*(valueSearch));  case *LESS\_THAN*:  if (isMatchesDateIsoFormat(valueSearch))  return builder.lessThan(root.get(criteria.getKey()), requestDate);  return builder.lessThan(root.get(criteria.getKey()), String.*valueOf*(valueSearch));  case *LESS\_THAN\_EQUALS*:  if (isMatchesDateIsoFormat(valueSearch))  return builder.lessThanOrEqualTo(root.get(criteria.getKey()), requestDate);  return builder.lessThanOrEqualTo(root.get(criteria.getKey()), String.*valueOf*(valueSearch));  case *LIKE*:  return builder.like(root.get(criteria.getKey()).as(String.class), String.*valueOf*(valueSearch));  case *STARTS\_WITH*:  return builder.like(root.get(criteria.getKey()).as(String.class), valueSearch + "%");  case *ENDS\_WITH*:  return builder.like(root.get(criteria.getKey()).as(String.class), "%" + valueSearch);  case *CONTAINS*:  return builder.like(root.get(criteria.getKey()).as(String.class), "%" + valueSearch + "%");  case *NOT\_LIKE*:  return builder.or(builder.notLike(root.get(criteria.getKey()).as(String.class), String.*valueOf*(valueSearch)), builder.isNull(root.get(criteria.getKey())));  case *NOT\_STARTS\_WITH*:  return builder.or(builder.notLike(root.get(criteria.getKey()).as(String.class), valueSearch + "%"), builder.isNull(root.get(criteria.getKey())));  case *NOT\_ENDS\_WITH*:  return builder.or(builder.notLike(root.get(criteria.getKey()).as(String.class), "%" + valueSearch), builder.isNull(root.get(criteria.getKey())));  case *NOT\_CONTAINS*:  return builder.or(builder.notLike(root.get(criteria.getKey()).as(String.class), "%" + valueSearch + "%"), builder.isNull(root.get(criteria.getKey()))) ;  default:  return null;  }  }  private Predicate getPredicate(CriteriaBuilder builder, Object valueSearch, Date requestDate, Join<T, Object> join) {  switch (criteria.getOperation()) {  case *EQUALITY*:  if ("null".equalsIgnoreCase(String.*valueOf*(valueSearch)))  return builder.isNull(join.get(criteria.getKey()));  if (isMatchesDateIsoFormat(valueSearch))  return builder.equal(join.get(criteria.getKey()), requestDate);  return builder.equal(join.get(criteria.getKey()), valueSearch);  case *NEGATION*:  if ("null".equalsIgnoreCase(String.*valueOf*(valueSearch)))  return builder.isNotNull(join.get(criteria.getKey()));  if (isMatchesDateIsoFormat(valueSearch))  return builder.notEqual(join.get(criteria.getKey()), requestDate);  return builder.notEqual(join.get(criteria.getKey()), valueSearch);  case *GREATER\_THAN*:  if (isMatchesDateIsoFormat(valueSearch))  return builder.greaterThan(join.get(criteria.getKey()), requestDate);  return builder.greaterThan(join.get(criteria.getKey()), String.*valueOf*(valueSearch));  case *GREATER\_THAN\_EQUALS*:  if (isMatchesDateIsoFormat(valueSearch))  return builder.greaterThanOrEqualTo(join.get(criteria.getKey()), requestDate);  return builder.greaterThanOrEqualTo(join.get(criteria.getKey()), String.*valueOf*(valueSearch));  case *LESS\_THAN*:  if (isMatchesDateIsoFormat(valueSearch))  return builder.lessThan(join.get(criteria.getKey()), requestDate);  return builder.lessThan(join.get(criteria.getKey()), String.*valueOf*(valueSearch));  case *LESS\_THAN\_EQUALS*:  if (isMatchesDateIsoFormat(valueSearch))  return builder.lessThanOrEqualTo(join.get(criteria.getKey()), requestDate);  return builder.lessThanOrEqualTo(join.get(criteria.getKey()), String.*valueOf*(valueSearch));  case *LIKE*:  return builder.like(join.get(criteria.getKey()).as(String.class), String.*valueOf*(valueSearch));  case *STARTS\_WITH*:  return builder.like(join.get(criteria.getKey()).as(String.class), valueSearch + "%");  case *ENDS\_WITH*:  return builder.like(join.get(criteria.getKey()).as(String.class), "%" + valueSearch);  case *CONTAINS*:  return builder.like(join.get(criteria.getKey()).as(String.class), "%" + valueSearch + "%");  default:  return null;  }  }  private boolean isMatchesDateIsoFormat(Object valueSearch) {  return String.*valueOf*(valueSearch).matches(*DATE\_ISO\_FORMAT\_REGEX\_FILTER*);  }  private void convertJsonPropToEntityField(Root<T> root) {  Class<?> cls = root.getJavaType();  Field[] declaredFields = root.getJavaType().getDeclaredFields();  while (cls.getSuperclass()!= null && !cls.getSuperclass().equals(Object.class)) {  cls = cls.getSuperclass();  declaredFields = ArrayUtil.*concatenate*(declaredFields, cls.getDeclaredFields());  }  for (Field field : declaredFields) {  if (field.isAnnotationPresent(JsonProperty.class) && field.getAnnotation(JsonProperty.class).value().equals(criteria.getKey())) {  criteria.setKey(field.getName());  }  }  }  private Date convertValueToDate(Object valueSearch) {  Date requestDate = null;  String requestDateTime = String.*valueOf*(valueSearch);  if (requestDateTime.matches(*DATE\_ISO\_FORMAT\_REGEX\_FILTER*)) {  Calendar calendar = DatatypeConverter.*parseDateTime*(requestDateTime);  requestDate = calendar.getTime();  }  return requestDate;  }  private Object convertValueToBoolean() {  Object valueSearch = criteria.getValue();  if (String.*valueOf*(valueSearch).equalsIgnoreCase(Boolean.*TRUE*.toString()) || String.*valueOf*(valueSearch).equalsIgnoreCase(Boolean.*FALSE*.toString()))  valueSearch = Boolean.*valueOf*(String.*valueOf*(valueSearch));  return valueSearch;  }  } |

## Test

Dùng CURL/POSTMAN để test

* Header cần truyền X-AUTH-TOKEN

|  |
| --- |
| curl  -X GET \ *{URL}/engineLogs?filter=engineDesc:\*"LCCF CE"\* AND sessionId:5 OR ( businessDate>"2018-11-15" AND id:401 )* \  -H 'cache-control: no-cache' \  -H 'content-type: application/json' \  -H 'X-AUTH-TOKEN: eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MCwidXNlcm5hbWUiOiJoYW50ODIiLCJzdWIiOiJzdWJqZWN0IiwiaXNzIjoiaXNzdWVyIiwiYXVkIjoiYXVkaWVuY2UiLCJleHAiOjE1ODY5NDgyNDUsIm5iZiI6MTU4NDM1NjIyNX0.vLVisLu5oY0I\_WjU9MhUCi8rLU4Bqctc5TgKw\_LizVS-nOv6BXTI8bYs1AvOAxj-k3hafZvlfBryAAEjoyGhnxM5R5EC24QghjiCl903H76YtzACLImvX2pULkhzL5pZEZeV5TDL8QnVghmJmseP4oC4LexEIEZ-0oM8uT41W24uZ4jDmBWH36\_R0FkxPw9DMks9Zcyni1d3Q0gWRc9\_paJscIpLtxu97rsXOBMTCtNTiiky-hYw0TZq59YqawfaxaDJO-dJTxP1f9pZ3v2PV4vaRDcFeGjUgBVx4cxnW7LDTb91VxxzV9UYYjWNZphyOifbJ7upIqW3FSM49VpQXA' \ |

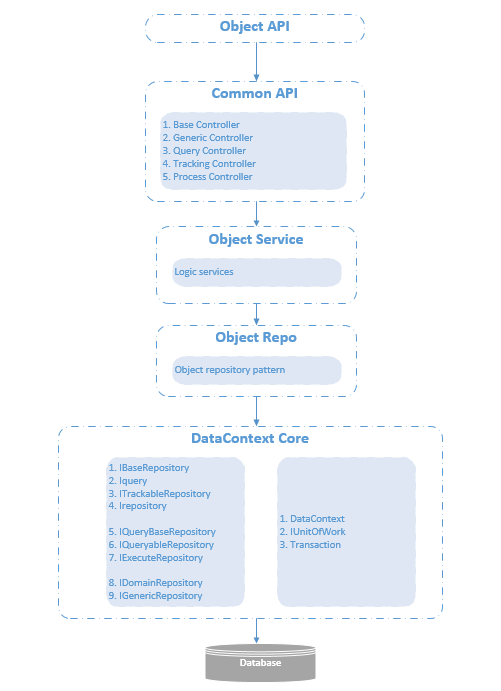
* Content-Type: Application/JSON

1. **.NET CORE GENERIC API**

## Overview

* Hệ thống IFRS9 làm việc dựa trên các Object được mapping 1:1 với các thực thể của SQL (Table, View, Function, Store Procedure). Do đó cần các generic API có chức năng truy xuất, tim kiếm, lọc, sắp xếp, thêm sửa xóa …
* Xuất phát từ yêu cầu trên, IT đã xây dựng framework đảm bảo cung cấp đủ các API generic, bao gồm:
  + Truy vấn data có phân trang, tìm kiếm dựa trên nhiều thông tin, sắp xếp
  + Chức năng thêm sửa xóa
  + Ghi audit log, user log
  + Notification

## Flow Chart

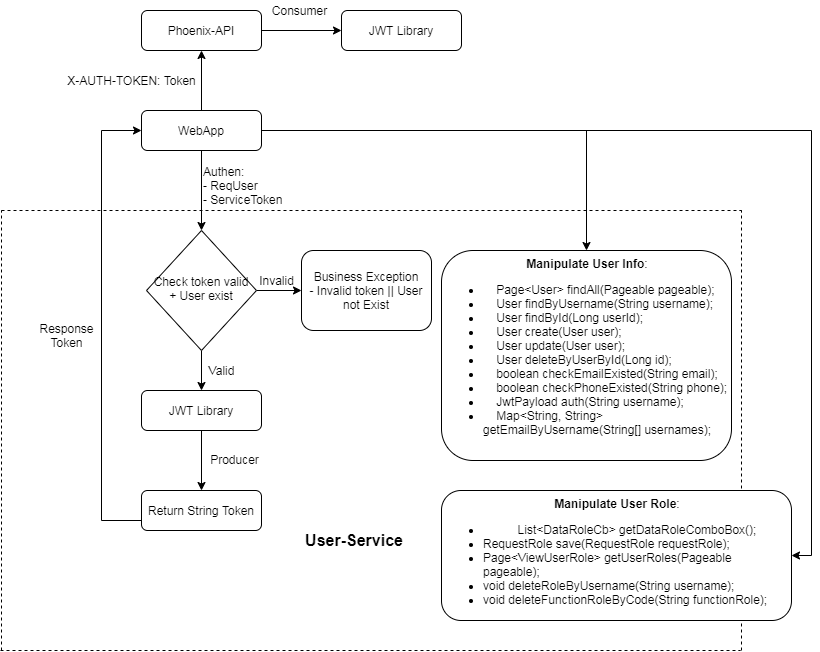


## Implementation

## Test

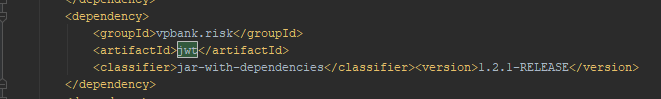
1. **USER SERVICES**
   * + - **Requirement & Purpose**

* Phoenix cần giao tiếp giữa các hệ thống với nhau. Các service gọi nhau, webapp tích hợp với API. Tất cả đều cần bảo mật và phân quyền.
* User-Service sinh ra để làm nhiệm vụ thao tác với thong tin của người dùng (users). Gồm việc login/thêm/sửa/xóa/phân quyền cho users. Sử dụng JWT để làm khóa giao tiếp giữa các hệ thống với nhau.
* JSON Web Token (JWT) là một chuẩn mở (RFC 7519) định nghĩa để truyền một cách an toàn thông tin giữa các bên dưới dạng đối tượng JSON. Thông tin này có thể được xác minh và đáng tin cậy vì nó có chứa chữ ký số. JWTs có thể được ký bằng một thuật toán bí mật hoặc một public/private key sử dụng mã hoá RSA. Phoenix đang sử dụng public/private key và lựa chọn jwt vì support đa ngôn ngữ, dễ tích hợp với các ngôn ngữ C#, Java…phù hợp với yêu cầu đưa ra
* Cả user-service và jwt library đều đã tách ra project riêng, không bị phụ thuộc vào phoenix và hiện tại chạy ổn định nên việc deploy hay phát triển mới của dự án đều ko bị ảnh hưởng
  + - * **Flow Chart**
  1. Workflow

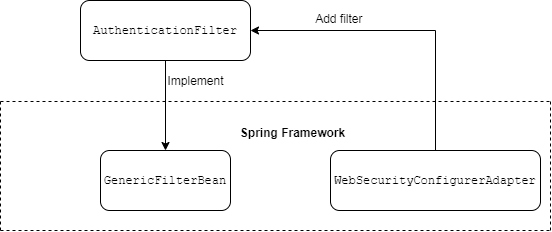


* Webapp tích hợp với User-Service. Khi user login vào hệ thống thì webapp sẽ gửi ReqUser(thông tin username) và 1 chuỗi ServiceToken để user-service đảm bảo service gọi đúng là nguồn từ webapp (cơ chế giống chữ ký điện tử).
* User-Service kiểm tra ServiceToken nếu sai hoặc user ko tồn tại trong hệ thống sẽ response về Business\_Exception cho client
* Nếu ServiceToken từ webapp gửi sang đúng và thông tin user đúng thì User-Service sẽ gọi jwt lib để generate ra chuỗi token. User login thành công
* Chuỗi Token response sẽ được sử dụng trong toàn bộ session login của user. Để kiểm tra phân quyền với mọi action của user.
* Phoenix-api: với mỗi lượt call endpoint bên phoenix-api thì sẽ webapp sẽ cần truyền vào header 1 custom-header là X-AUTH-TOKEN được quy định trước giữa 2 hệ thống. Giá trị của header này là chuỗi token response sau khi user login thành công
* Ngoài ra để phục vụ các chức năng thay đổi thông tin user mà được lưu data ở Auth database thì webapp cũng cần phải tích hợp để gọi các endpoint của user-service (find user by username, create user, change function/data role for user…)

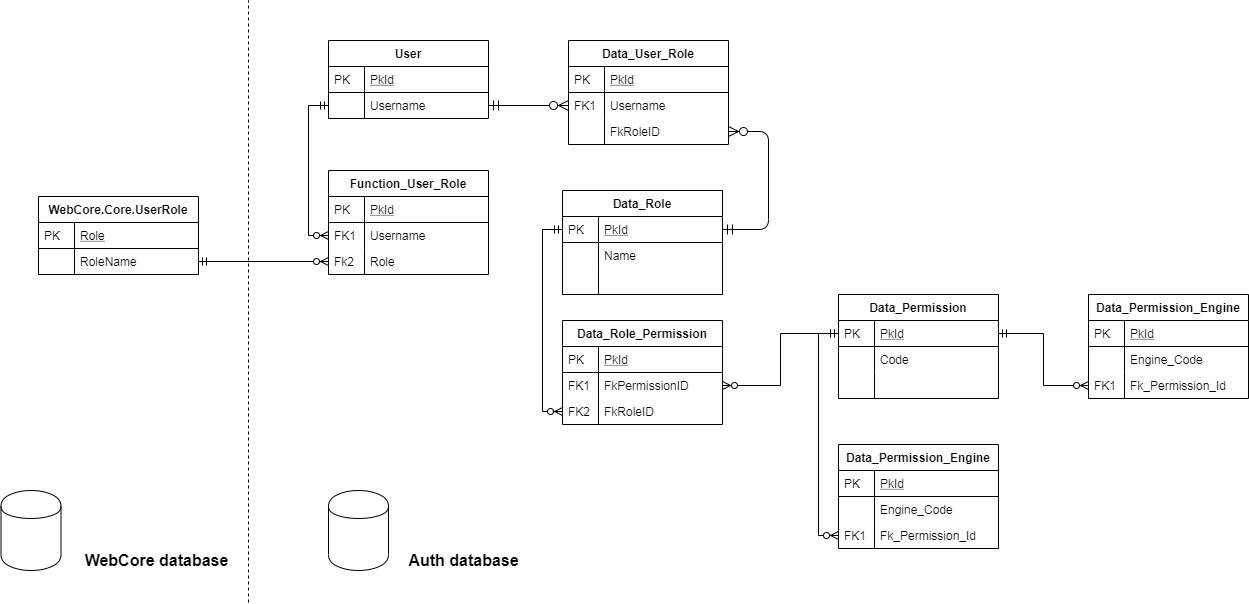
* 1. *JWT Library*
* JWT là thư viện được dùng chung giữa các service với nhau. Theo chuẩn jwt gồm 3 phần: header.payload.signature
  + Header bao gồm hai phần chính: loại token (mặc định là JWT – Thông tin này cho biết đây là một Token JWT) và thuật toán đã dùng để mã hóa (hiện đang dùng thuật toán RSA).
  + Payload chứa các claims. Claims lưu thông tin User và một số metadata. Có 3 loại claims thường gặp trong Payload: reserved, public và private claims. Phoenix chỉ sử dụng Reserved và Private
    - Reserved claims: Đây là một số metadata bắt buộc, số còn lại nên tuân theo để JWT hợp lệ và đầy đủ hong tin: iss (issuer), exp (expiration time), sub (subject), aud (audience). Hiện đang để cấu hình trong application.properties của từng service
    - Private Claims – Claims tự định nghĩa được tạo ra để chia sẻ hong tin giữa 2 services với nhau như function\_role, data\_role, is\_admin
* Project jwt là 1 project ổn định và đã được tách ra ngoài làm thư viện riêng. Hiện đang chạy version 1.2.1 release. Sử dụng maven để build ra file .jar vào thư mục m2(maven) trên máy hoặc trên các repository private/public. Các project khác có thể chạy cùng và phải khai báo dependency
* Jwt expose ra 2 hàm chính là producer (tạo token) và consumer(verify token)



* 1. *Phoenix-api tích hợp để sử dụng token như thế nào*



* AuthenticationFilter sử dụng interface GenericFilterBean có sẵn của Spring Framework để filter các http request trước khi vào Controller. Class này đọc chuỗi token, kiểm tra token valid và gán lại function role/data role
* Trong cấu hình của project thêm AuthenticationFilter làm mặc định filter http request đến
* Ngoài ra có định nghĩa thêm vài @Bean để custom message lỗi khi user bị access denied cho webapp là RestAccessDeniedHandler và RestAuthenticationEntryPoint
  1. *Thiết kế Database trong Auth cho role*



* + - * **How to use**

*Xem sử dụng trong Swagger-UI*

*{URL}:1111/swagger-ui.html*

* + - * **Implementation**

Source code trên TFS tại: <http://10.37.16.226:86/DefaultCollection/IRS/_git/User-API>

* + - * **Test**

Sử dụng try it out với swagger hoặc dùng postman

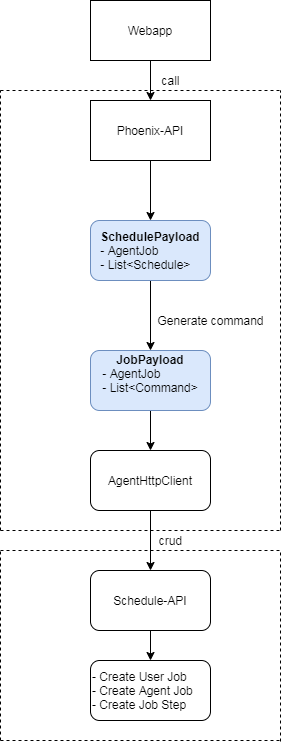
1. **SCHEDULE SERVICES**

**Requirement & Purpose**

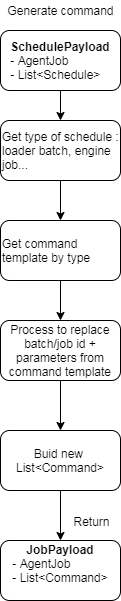
* Schedule-Service là service để hỗ trợ việc đặt lịch chạy cho Job/Batch.
* Schedule-Service được thiết kế để không quan tâm item cần chạy là job/batch hay engine. Client tích hợp cần gửi thông tin là command/script cần execute và thời gian chạy, lịch chạy (one time hoặc recurrent…). Có thể mở rộng sau này đặt lịch những phần khác ngoài batch/job
* Schedule làm cơ chế đẩy lệnh vào SQL Agent Job nên có hạn chế chỉ support SQL Server ở thời điểm hiện tại nhưng bù lại việc đẩy lệnh vào SQL Agent Job quản lý khá ổn định
* Có thể tích hợp cho các project khác cần sử dụng việc đặt lịch
* Đã tách ra project riêng, không bị phụ thuộc vào phoenix và hiện tại chạy ổn định nên việc deploy hay phát triển mới của dự án đều ko bị ảnh hưởng

**Flow Chart**

* 1. *Workflow*



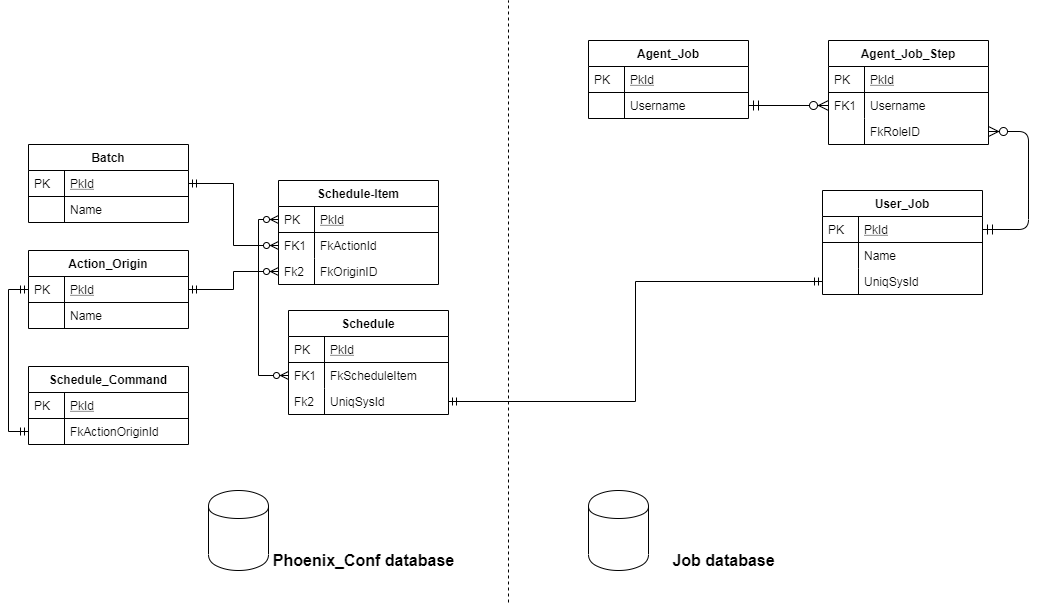
* Webapp call phoenix-api. Gửi sang thông tin SchedulePayload :
* AgenJob: Là thông tin về đặt lịch, đang sử dụng theo thông số được định nghĩa chuẩn của SQL Agent Job
* List<Schedule>: danh sách của schedule-item (VD: Kiểu chạy là batch-loader và id của batch là 62…) và thứ tự chạy của từng schedule-item
* Phoenix-api nhận input SchedulePayload để generate ra command theo template mẫu (bảng Phoenix\_Conf.App.Schedule\_Command) đã được định nghĩa với từng loại schedule-item (batch/job). Trả ra output là JobPayload. JobPayload này sử dụng để giao tiếp với Schedule-api.
* Phoenix-api sinh ra 1 unique-system-id sử dụng để làm khóa với data với bên schedule-api
* AgentJobHttp là internal service trong phoenix-api để giao tiếp với schedule-api. Hiện tại đang tích hợp với schedule-api để thao tác thêm/sửa/xóa lịch chạy



Cơ chế generate command

*\*\* Note: Việc thêm phần xử lý để generate ra command vì thiết kế schedule-api không cần quan tâm đến nghiệp vụ xử lý của client(phoenix-api) mà chỉ quan tâm thông tin đặt lịch và command cần được trigger. Phoenix-api thì ko cần quan tâm đến việc setup lịch mà chỉ call qua schedule-api*

* Schedule-api call SP Phoenix.app.Manage\_Job để thực hiện việc create/delete/update agent job
* Schedule đang được viết bằng ngôn ngữ Scala. Project hiện tại vẫn hỗ trợ thêm code Java chạy cùng nên sẽ không ảnh hưởng việc maintain
  1. *Thiết kế Database cho phần schedule*



* Đường nối giữa bảng Schedule và User\_Job thực chất **không phải** là khóa ngoại trong database mà dùng để đánh dấu item-id giữa 2 hệ thống với nhau. Trường hợp sau này có tích hợp có lỗi thì vẫn có trường item-id để trace lại nếu có lỗi

**How to use**

*Xem sử dụng trong Swagger-UI*

*{URL}:8888/swagger-ui.html*

**Implementation**

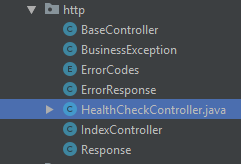
Source code trên TFS tại: <http://10.37.16.226:86/DefaultCollection/IRS/_git/Schedule-API>

**Test**

Sử dụng try it out với swagger hoặc dùng postman

1. **BASECONTROLLER + HTTP PACKAGE**
   * + 1. **Requirement & Purpose**

* BaseController là class mà các Controller trong Phoenix đều extend.
* HTTP là package cấu hình và điều phối http request cho Phoenix: gồm handler exception, health check, return customized response…
  + - 1. **Flow Chart**



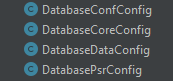
* Nhiệm vụ của BaseController là handler Exception cho 2 trường hợp, trả ra đúng BusinessException.
  + Handler Runtime Exception
  + Handler Validate Exception để đẩy ra đúng customized message
* Trong phoenix-api có BusinessException được định nghĩa để routing tất cả các Exception phải trả về object này và có các thuộc tính
  + Code: Mã lỗi được định nghĩa trong ErrorCodes
  + Description: Mô tả lỗi được định nghĩa trong message.properties
  + HttpStatus.
* HealthCheckController là endpoint được truy cập bởi {URL}/health. Dùng để kiểm tra trạng thái nhanh của các 3rd parties đang sử dụng như (SQL database connection, elasticsearch, rabbitmq, user-service…)
* IndexController hiện dùng để hiển thị cho client biết version hiện tại của API
* Response là empty entity, hỗ trợ việc return kiểu empty cho Controller. Do Controller trong Spring framework không cho phép trả về empty nên sử dụng Response để wrap empty

1. **MULTIPLE DATABASE CONNECTIONS**

**Requirement & Purpose**

* Phoenix hiện tại đang kết nối tới 4 database: Phoenix\_Conf, Phoenix\_Data, PSR, Core vì:
  + Liên kết chặt chẽ giữa 2 database Phoenix\_Conf và Phoenix\_Data nên trong phoenix-api vẫn cần kết nối đến cả 2 database để cùng lấy dữ liệu.
  + Ngoài ra việc cần phát triển nhanh trong thời gian ngắn với database PSR cũng không tách ra thành service riêng được nên phoenix-api cũng đang kết nối tới database PSR
  + Manual file hiện tại cũng đang cần thông tin từ database Core của Datamodel

**Flow Chart**



* Mỗi connection được quản lý bởi 1 class Config của Spring, tương ứng với mỗi connection sẽ có từng entityManager với unit tương ứng. VD:

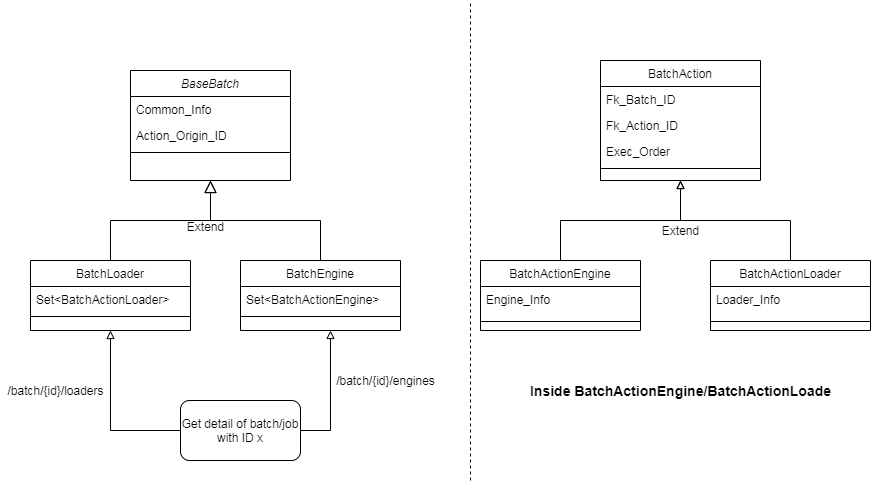
@PersistenceContext(unitName = "phoenixConf")  
private EntityManager entityManager;

Được sử dụng để connect đến database Phoenix\_Conf. Thông số kết nối đến các database nằm trong file application.properties. Khi sử dụng biến này cần phải khai báo rõ unitName, mặc định sẽ trỏ đến database Phoenix\_Conf.

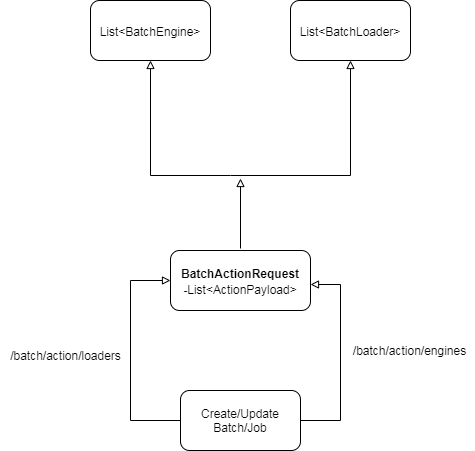
* Spring framework hỗ trợ việc cấu hình database connect đến database nào theo package định sẵn. Hiện tại dev đã tách các package tương ứng đến connection của databse. VD connection đến PSR database sẽ được áp dụng với toàn bộ repository/model của package psr.

1. **SMT- LOADERS/ENGINES & BATCH/JOB**
   * + 1. **Requirement & Purpose**

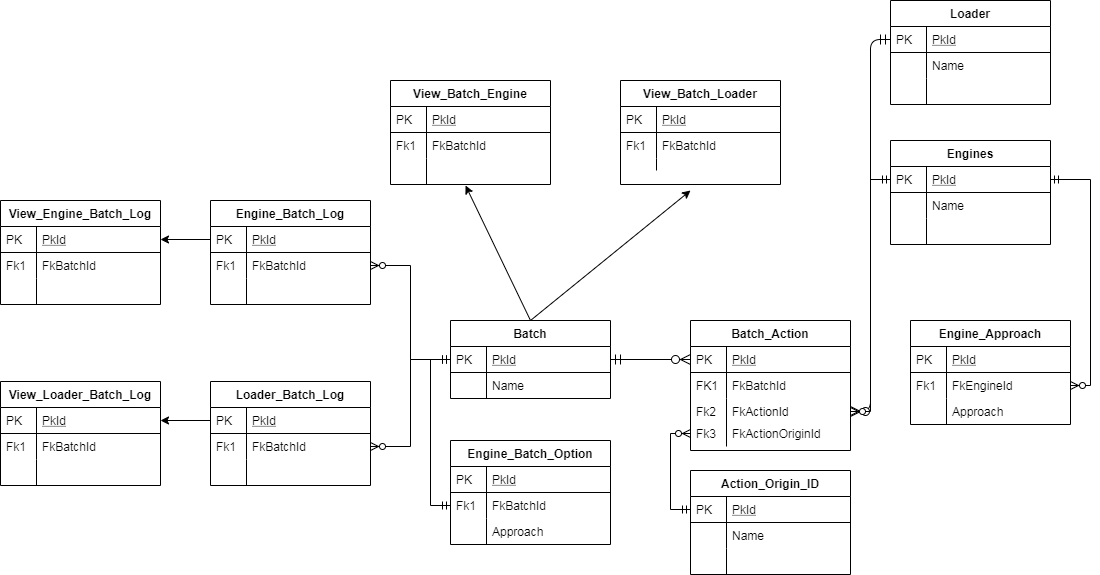
* Phoenix có 1 module là System Management Tools. Dùng để chạy Loader/Engine và các Batch/Job.
* Thiết kế để Batch/Job có thể mở rộng ra sau này không chỉ chạy cho mỗi Loader/Engine. Loader/Engine đều có những thông tin khác nhau nên sẽ cần đưa ra 1 layer đứng giữa, đặt tên là Batch
  + - 1. **Flow Chart**
  1. *Lấy thông tin chi tiết của Batch Loader hoặc Engine Job*



* Webapp call phoenix-api. Gửi sang thông tin ID của Batch/Job :
* Nếu cần lấy thông tin batch loader sẽ gọi sang : /batch/{id}/loaders.
* Nếu cần lấy thông tin engine job sẽ gọi sang : /batch/{id}/engines.
* Tùy theo request gửi sang là gì thì api sẽ điều hướng đến internal service bên trong để trả ra BatchLoader hoặc BatchEngine
* Trong đó thì BatchLoader có chứa thông tin là 1 danh sách unique của BatchActionLoader, Được kế thừa từ BaseBatch. BaseBatch là class chưa thông tin chung như batch\_name, action\_origin\_id…
* Các thuộc tính bên trong của BatchActionLoader là thông tin của từng loader. Ngoài ra BatchActionLoader kế thừa từ BatchAction (data trong bảng Batch\_Action) để nhận thông tin như execute\_order, id của loader…)
* Tương tự với nhánh của BatchEngine cũng xử lý giống cơ chế như thế
  1. *Thêm mới hoặc sửa Batch/Job*



* Webapp call phoenix-api theo routing url được định nghĩa. /batch/action/engines hoặc /batch/action/loaders với request body là BatchActionRequest chứa danh sách của ActionPayload.
* Service trong api xử lý và trả ra danh sách của BatchEngine/BatchLoader tùy theo routing
  1. *Thiết kế Database cho phần Batch/Job*



* Engine cấu hình có link đến table Engine\_Approach theo yêu cầu của nghiệp vụ
* API hiện sử dụng các views để hiển thị các pages về engines/batch/job/loader và các pages monitor logs. Mục đích sử dụng views là vì :
* Cần lấy thêm thông tin chi tiết của từng engine/load từ BatchID;
* Cần trường thông tin về engine/entity cụ thể để kiểm tra data roles
* Với thiết kế cho phép mở rông như hiện tại thì Extractor/BatchExtractor cơ chế gắn vào Batch tương tự như BatchLoader/EngineJob.

1. **How to use**

*Xem sử dụng trong Swagger-UI*

*{URL}:8096/swagger-ui.html*

1. **Implementation**

Source code trên TFS tại: <http://10.37.16.226:86/DefaultCollection/IRS/_git/Phoenix-API>

1. **Test**

Sử dụng try it out với swagger hoặc dùng postman

1. **ADDITIONAL**
2. **Requirement & Purpose**

* Phần này sẽ nói về caching, cơ chế async, generic api, cơ chế validate…, một số cấu hình trong project, không liên quan nhiều đến nghiệp vụ của Phoenix

1. **Details**

*2.1.* Caching trong Phoenix-api

* Caching của phoenix-api sử dụng interface SimpleCacheManager của Spring framework. Sử dụng để lưu 1 số data nặng hoặc xử lý tốn nhiều thời gian. Data được khởi tạo từ khi start service và đưa vào in-memory.
* Hiện tại có cache 4 phần là:

new ConcurrentMapCache("actionOrigin"),  
new ConcurrentMapCache("derivedFieldsComboBox"),  
new ConcurrentMapCache("informationSchemas"),  
new ConcurrentMapCache("mpScenarioVersion")

* Tất cả cache đều được định nghĩa trong interface InMemoryCache
* Trước khi sử dụng cần khai báo trong package \*/config/CachingConfig và khởi tạo mới 1 biến ConcurrentMapCache
* Phoenix-api có khai báo 1 component là PhoenixContext, được trigger chạy mỗi khi start service bởi StartupListener. Event này chạy async với việc khởi động service. Nhiệm vụ của PhoenixContext là khởi tạo data và đưa vào in-memory

*\*\*Note: Hiện tại cache đang chưa có phần reload cache, dữ liệu mới vào thì phoenix-api sẽ chưa hiển thị mà cần restart service. Cần phải làm endpoint để có thể thực hiện hot reload cache*

### *2.2. Common Lib*

* CommonRepository hiện đang có 2 hàm để lấy ngày làm việc hiện tại của hệ thống (currentWorkingDate) và hàm lấy getWorkingDate(Date inputDate, int range) theo range.
* AliasToEntityOrderedMapResultTransformer: Trả ra kết quả từ function table / procedure từ SQL theo kiểu HashMap và được sắp xếp đúng thứ tự như SQL hiển thị
* ArrayUtil: Các hàm Util xử lý cho Array
* BelongObject: Xử lý cho generic filter, kiểm tra xem field X có phải là khóa ngoại của 1 table khác không
* ClassUtil: Dùng cho reflection
* DateTimeUtil:
* ExcelUtil
* FTPUtil
* JsonUtil
* ListUtil
* NumberUtil
* ReflectionUtil
* Serializer: Parse các object ra byte[] và convert ngược lại
* StringUtil
* XLSX2PrintStream: Xử lý đọc file excel khi import file

### *2.3. Bidirectional và Unidirectional*

*\*\* Note: Dev nên đọc cái này trên google để hiểu về principle của nó: https://stackoverflow.com/questions/5360795/what-is-the-difference-between-unidirectional-and-bidirectional-jpa-and-hibernat*

* Để hiểu đơn giản nó giống như việc khai báo 1-n, n-n trong SQL của Object
* Các kiểu quan hệ @OneToMany và @ManyToOne @OneToOne, @ManyToMany
* Hibernate hỗ trợ việc trả ra các object child/parent nếu khai báo đủ mối quan hệ
* Vấn đề hiện tại: Thời gian đầu áp dụng vào project thì không được hiệu quả vì sau khi khai báo, framework tự generate khá nhiều câu lệch SQL nested cho việc lấy data đơn giản, chia nhỏ câu query, hit lên databse khá nhiều. Khó để trả đúng response cho client vì response trả luôn ra nested-object. Giai đoạn sau đã thay đổi khi cần lấy data có quan hệ phức tạp bằng tạo views dưới Database và API lấy trực tiếp qua views. Điểm yếu của việc sử dụng qua views là không index được, nếu dữ liệu lớn hơn thì cần tuning kiểu khác.
* Các phần đang sử dụng Bidirectional và Unidirectional chỉ có phần System Management Tools. Phần monitor logs đã không còn sử dụng.

### *2.4 . Asynchronous Thread*

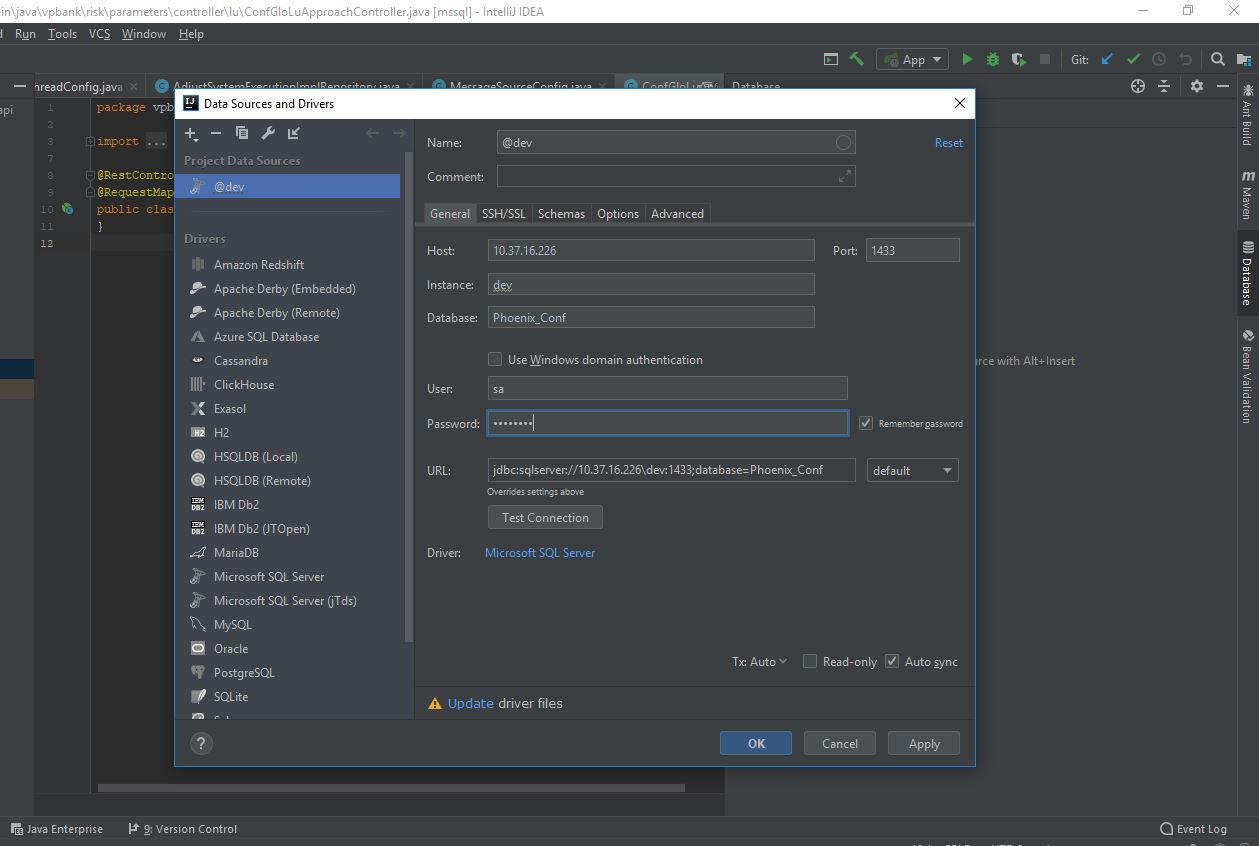
* Các endpoint cần đợi Procedures xử lý lâu hoặc phải trả về cho client ngày thì sẽ chạy qua Async-Thread
* Hiện tại đang áp dụng với các phần execute engines/loaders/archive. Ngoài ra còn có Audit cũng sử dụng async-thread vì audit cần chạy ngầm ở dưới background, nếu có exception trong lúc ghi log audit cũng không ảnh hưởng đến feature đang hoạt động. Việc gửi Noti cũng chạy Async tương tự như Audit
* Được cấu hình trong package \*/config/ThreadConfig
* Các hàm sử dụng Async đều cần khai báo annotation @Async("tên\_thread\_pool"). Kiểu trả về bắt buộc là kiểu Future/ CompletableFuture để có thể xử lý data tiếp sau khi hàm chạy xong.

### *2.5. Cấu hình message sources cho thông báo lỗi*

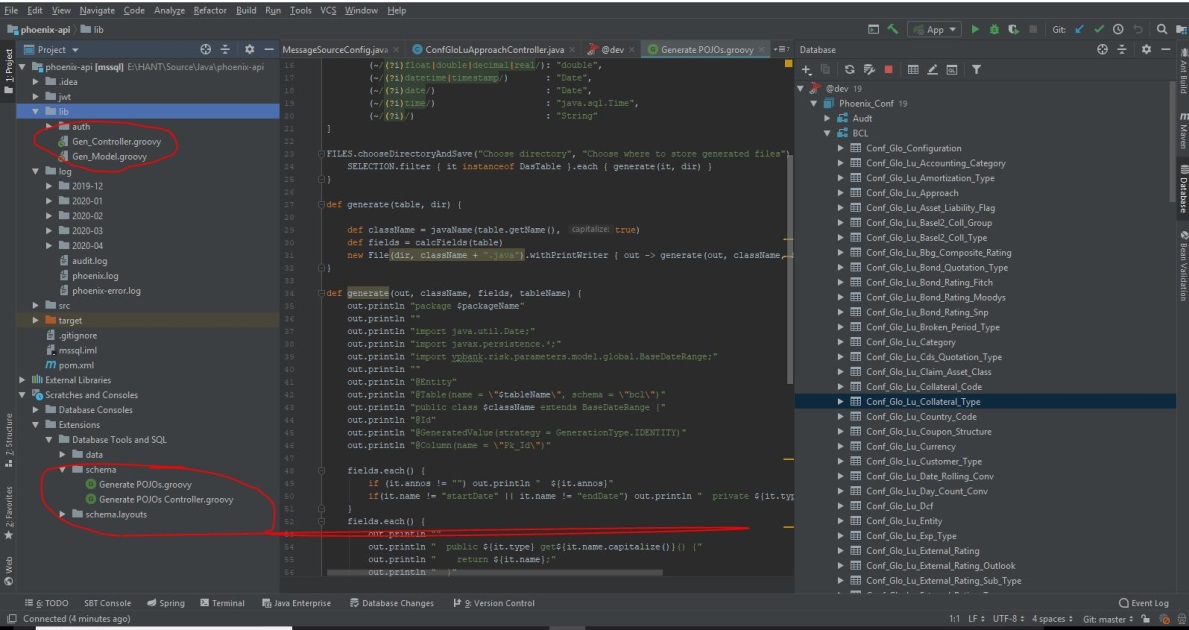
* Sử dụng mặc định message sources của framework. Hỗ trợ đa ngôn ngữ
* Khai báo trong package \*/config/ MessageSourceConfig
* Nội dung của message nằm trong /resources/message.properties
* Thông tin của từng message cần mapping đúng với BusinessException và ErrorCodes. Key của message là description trong ErrorCodes

### *2.6. Generic api cho parameters*

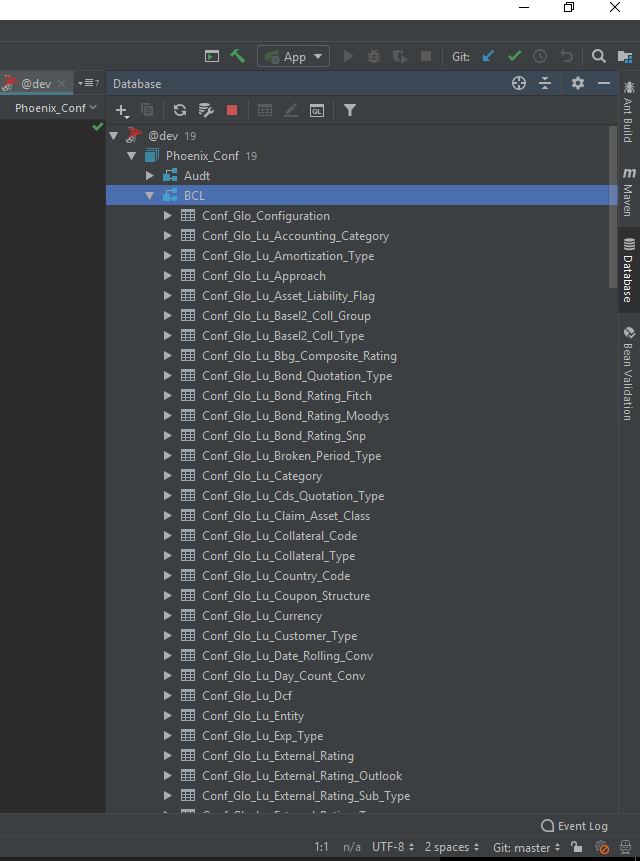
* Hiện tại có nhiều pages CRUD đơn giản và có logic giống nhau nên phoenix-api có 3 class hỗ trợ cho việc tạo generic-endpoint:
  + GenericRepository: xử lý CRUD, hỗ trợ cả generic filter với mỗi object T là kiểu generic
  + GenericService: xử lý logic chung, giống nhau của các api
  + GenericController: được extend từ BaseController. Đảm bảo các generic- endpoint đều phải có syntax sources giống nhau. VD:
    - Lấy danh sách là GET-conf/lu/approach,
    - Thêm mới là POST- conf/lu/approach
    - Sửa là PUT- conf/lu/approach
* Việc này sẽ tiết kiệm rất nhiều thời gian khi tạo 1 pages mới
* Khi cần tạo 1 endpoint của 1 tables mới thì chỉ cần:
  + Khai báo (model) Object mapping với table trong database
* @Entity  
  @Table(name = "Conf\_Glo\_LU\_Approach", schema = "bcl")  
  public class ConfGloLuApproach extends BaseDateRange {  
   @Id  
   @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
   @Column(name = "Pk\_Id")  
   private Long id;  
   private String approachCode;  
   private String approachName;  
    
   public Long getId() {  
   return id;  
   }  
    
   public void setId(Long id) {  
   this.id = id;  
   }  
    
   public String getApproachCode() {  
   return approachCode;  
   }  
    
   public void setApproachCode(String approachCode) {  
   this.approachCode = approachCode;  
   }  
    
    
   public String getApproachName() {  
   return approachName;  
   }  
    
   public void setApproachName(String approachName) {  
   this.approachName = approachName;  
   }  
    
   @Override  
   public String toString() {  
   return "ConfGloLuApproach{" +  
   "id=" + id +  
   ", approachCode='" + approachCode + '\'' +  
   ", approachName='" + approachName + '\'' +  
   "} " + super.toString();  
   }  
  }
  + Khai báo (controller) ObjectController để có RequestMapping mặc định
* @RestController  
  @RequestMapping(value = "conf/lu/approach")  
  public class ConfGloLuApproachController extends GenericController<ConfGloLuApproach> {  
  }
* Trong VD trên thì Object kế thừa BaseDateRange. Phoenix-api có một số class Base để khai báo các cột chung mà nhiều bảng sử dụng như: start\_date, end\_date, approach, scenario, entity, version… Việc kế thừa thuộc tính của Base class giúp việc thay đổi tên khi database thay đổi sẽ nhanh hơn. Không phải maintain ở từng object
* Các class generic nhưng vẫn hỗ trợ việc override logic riêng của từng endpoint. Xem VD tại ConfCrLookupGuiController, ngoài những endpoint mặc định thì vẫn cho phép mở rộng nếu cần
  1. *Generate Code cho các generic-endpoint*
* Phoenix-api có nhiều bảng lookup-mapping. Để tạo đủ pages thì cũng cần phải tạo số Model + Controller tương ứng. Việc tạo tay dễ sai và mất công vì phải đặt tên biến và data types theo convention của Object
* Module hỗ trợ code được viết bằng ngôn ngữ Groovy vì intellji support môi trường sẵn
* Hướng dẫn sử dụng:
* Sử dụng Database Tools của Intellij để kết nối với database



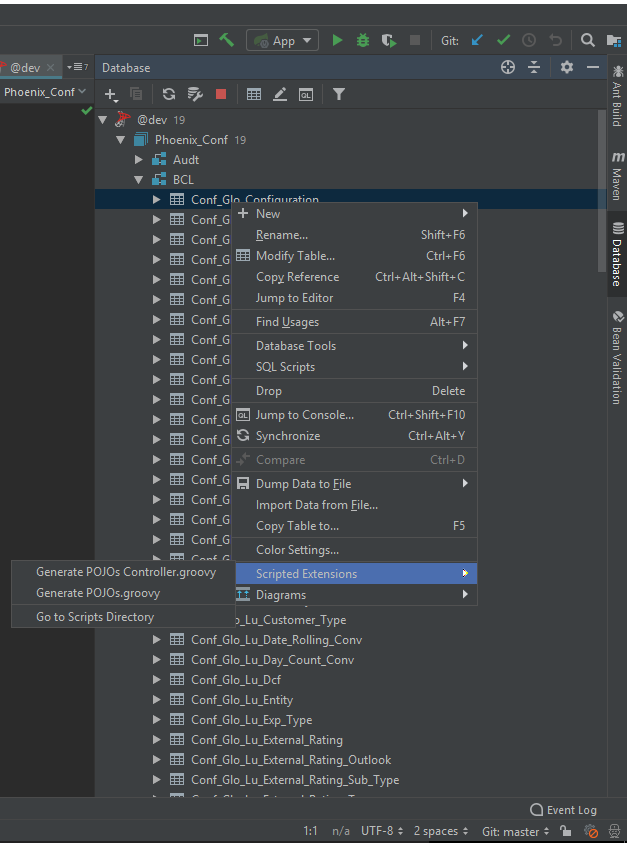
* Ghi đè script generate của Intellji bằng script trong phoenix-api/lib/Gen\_Controller.Groovy và phoenix-api/lib.Gen\_Model.Groovy. 2 Script này được viết sẵn để customize cho phoenix



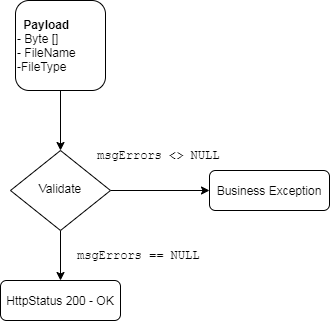
* Chọn table cần generate ra Model/Controller



* Chọn Generate Controller/Table. Chọn Output Folder là địa chỉ trên máy tính



* Copy file từ output vào trong project. Rebuild project.
  1. *Import cho CfAssumptions*
* Import file excel cho CashFlow Assumption do nhiều dữ liệu và sử dụng file excel. Requirement cũng có nhiều step kiểm tra validate đặc thù nên phải làm module riêng
* Sau khi chuyển sang tính năng request – approval thì webapp cần call 2 lần API.
  + Lần 1 để gửi file sang validate file trước khi import vào
  + Lần 2 gửi sang 1 request với thông tin đã upload file lên thư mục tạm trên server, API cần đọc file từ địa chỉ thư mục tạm, ghi xuống database.



**Lần 1 gửi sang validate**

* Class thực hiện validate là ImportValidate, được tách ra ngoài để có thể sử dụng chung cho các file import
* Validate file cần kiểm tra data types của từng value trong mỗi cell có khớp với database structure không. Kiểm tra ràng buộc giữa 1 số business logic như cặp scenario-version; approach phải trong danh sách định nghĩa sẵn…
* Nếu validate không thành công thì API sẽ trả về danh sách lỗi định dạng: *versionId: Version\_ID and Scenario\_ID are not applicable*. Nếu có nhiều lỗi sẽ được cách nhau bới dấu pipe **|**
* Webapp đọc mã lỗi và đưa vào 1 cột Error để users có thể biết lỗi và cập nhật lại file
* Nếu validate thành công thì API trả về Response empty. Webapp thực hiện upload file lên folder trên server thông qua FTP và tạo 1 “request for approval” mới gửi sang cho API
* API đọc file từ FTP, xóa hết dữ liệu trong bảng theo quyền của users và thực hiện ghi dữ liệu vào database
  1. *Audit*
* Là common function (BaseController), build file audit.log với cấu trúc message dạng, Time\_Type\_User\_Action\_PayloadData

Audit.log files

Audit Function

BaseController

Logstash service

Kibana (UI)

Elastic search

* 1. *Manual upload file*

Files trong request folder (WebCore upload files), được di chuyển qua các target folder bằng thư viện FTP Client (Manual File Handler)

Config trong bảng Phoenix\_Conf.dbo.FTP\_Config



FTP Client

Files (DM server)

Files (DM server)

Files (PHX server)

*\*\* Note: Hiện tại framework bắt buộc phải sử dụng Pk\_ID là sequence thì mới cho phép Bulk Insert/Update nên nếu file nhiều dữ liệu (> 15-20mb) thì sẽ phải đợi lâu hơn. Cơ chế hiện tại không support bulk insert/update*

1. **How to use**

*Xem sử dụng trong Swagger-UI*

*{URL}:8096/swagger-ui.html*

1. **Implementation**

Source code trên TFS tại: <http://10.37.16.226:86/DefaultCollection/IRS/_git/Phoenix-API>

1. **Test**

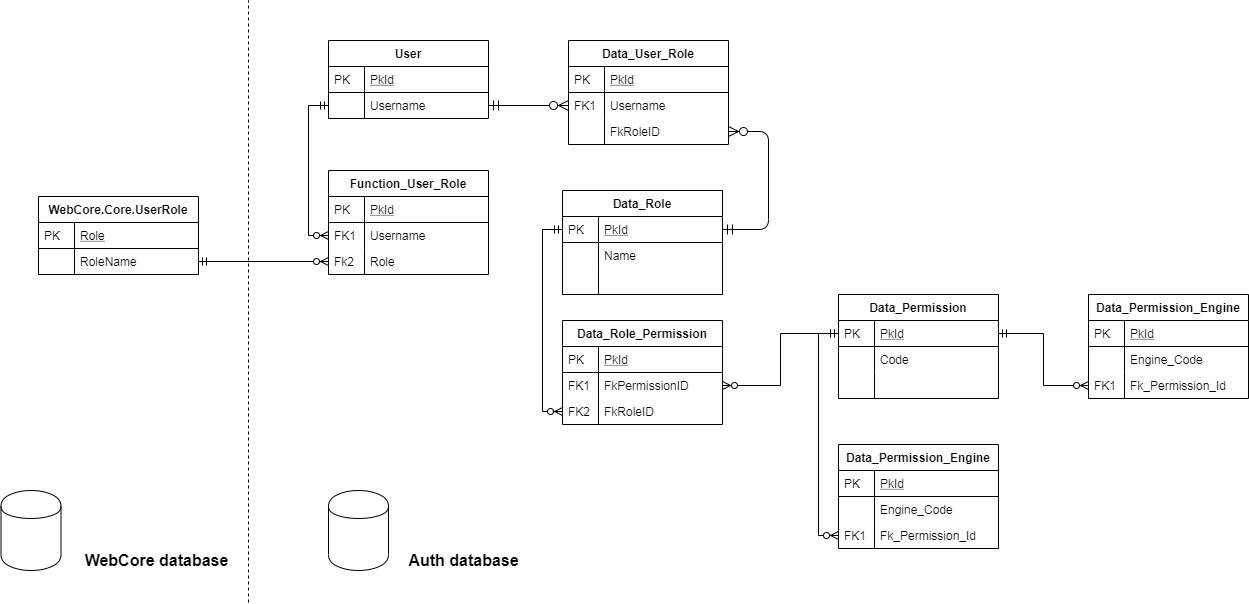
Sử dụng try it out với swagger hoặc dùng postman

1. **Data roles**
   * + 1. **Requirement & Purpose**

* Phoenix phân quyền ở mức data. Yêu cầu cụ thể là theo colums engine\_code và entity\_code. Design hiện tại cho data roles không bị cố định ở 2 cột này mà cho phép mở rộng theo các chiều khác ngoài engine\_code và entity.
* Phân quyền data roles chia nhỏ thành các permission. Các permission gắn với các key của bảng nghiệp vụ. VD Permission\_Engine thì gắn với engine\_code
* User được cấp quyền qua roles chứ không trực tiếp gắn vào permission
* Mỗi roles có thể có nhiều permission

\*\* Note: Hiện tại hệ thống chưa hỗ trợ phân quyền cho Entity Admin.

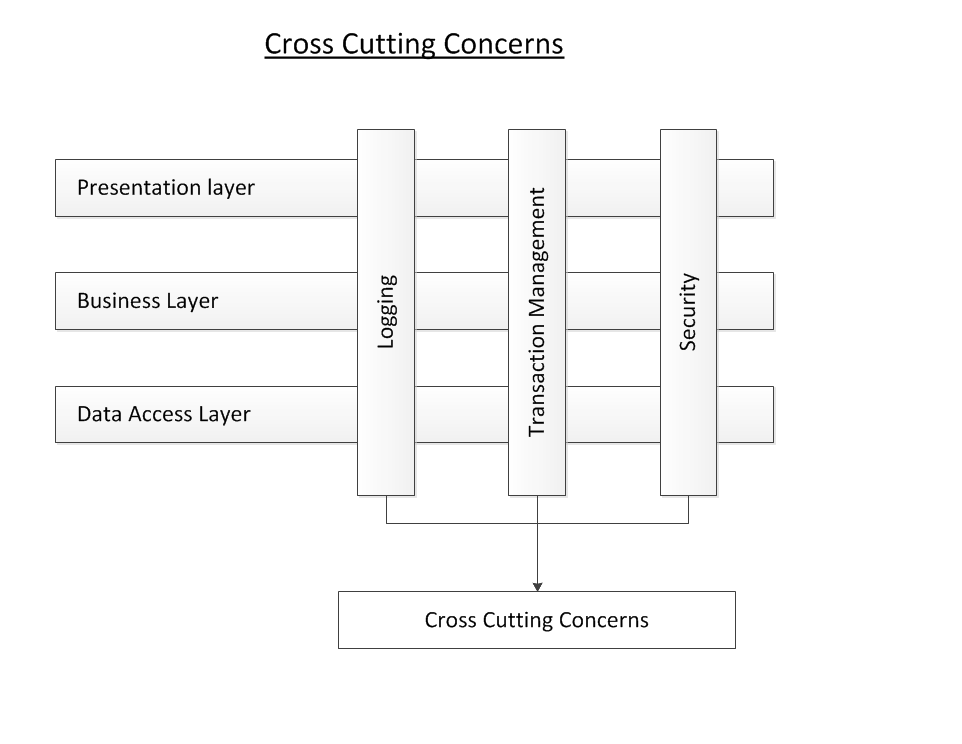
* + - 1. **Thiết kế database phần roles**



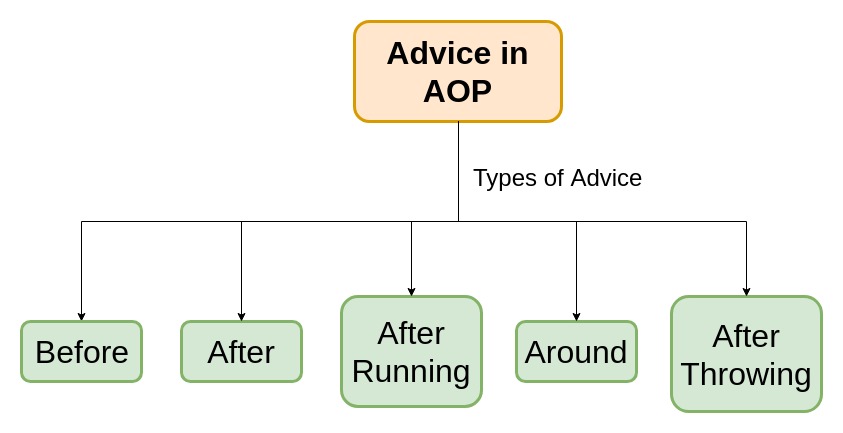
* Các permission của function\_roles được lưu tại database WebCore. Nguyên nhân do function roles có nhiều permission. Gọi qua http chậm, permission về function roles cũng thuộc logic của webapp.
* Quan hệ giữa user-function roles được lưu tại database Auth. Mục đích để trả kèm token lúc giao tiếp và lưu dữ liệu về quyền người dùng tập trung tại 1 database

1. **AOP - aspect-oriented programming**

* Data roles trong phoenix sử dụng mô hình lập trình (programming paradigm) là AOP - aspect-oriented programming.
* AOP hỗ trợ việc tạo 1 service làm cutting giữa toàn hệ thống phoenix-api.



* 1 số khái niệm của AOP:
* Core concerns: hàm chính của chương trình (các method cần thực hiện log)
* Join points: một điểm của service, là nơi có thể chèn những “custom action”
* Pointcut
* Advice: là những xử lý phụ được thêm vào xử lý chính, code để thực hiện các xử lý đó được gọi Adivce.
* Các kiểu của Advice:



* Phoenix-api có 1 @Component là **EnableFilterAspect** sử dụng để chèn thêm các action tại các poincut trong hệ thống. Các action này sẽ kiểm tra thông tin về quyền user hiện tại để thay đổi SQL query trước khi execute

*\*\* Note: Vấn đề hiện tại là Spring có nhiều interface cho phép call lệnh SQL nên phải customize tất cả poincut trong các interface của framework*

* Hiện tại đã customize được các interface:
  + CRUDRepository: findByXXX, saveAll, save, countByXXX, findById, findByXAndYNotIn, findByDistinc…Với X, Y là column bất kỳ của bảng. Tất cả các câu query qua interface này đều được kiểm tra role và thêm điều kiện để lấy data đúng quyền người dùng (Build ra kiểu \*and engine\_code in(roles)). Đảm bảo không bị ảnh hưởng các điều kiện AND OR của câu query gốc.
  + Tất cả các lệnh được viết bằng native query – raw SQL mà không qua object. Được xử lý tại **EnableFilterAspect**

@AfterReturning(  
 pointcut = "bean(entityManagerFactory) && execution(\* createEntityManager(..) )",  
 returning = "retVal")  
public void getSessionAfter(JoinPoint joinPoint, Object retVal)

* + Xử lý riêng cho lấy data với Batch/Job/ScheduleItem vì những object này không có thông tin thẳng đến engine\_code ( xem code tại **EnableFilterAspect).** Hiện tại đang phải dùng 1 số Reflection nên cũng khó debug hơn.
* Cần khai báo annotation @Filter tại các Object có các cột engine\_code và entity\_code. VD trong package/\*/ ViewEngine.java

@Filter(name = *ENGINE\_ROLE*)  
@Filter(name = *ENTITY\_ROLE*)

* Khai báo trong file package.java chi tiết nội dung của @Filter, để không phải khai báo chi tiết cho các object trong cùng package. (Xem thêm file package.java)
* Thiết kế AOP giúp việc turn on hay turn off kiểm tra quyền sẽ dễ dàng hơn. Muốn turn off chỉ cần comment 2 dòng code trên đầu class. Spring framework sẽ không scan đến **EnableFilterAspect.** Thiết kế này giúp cho không phải sửa code trên toàn hệ thống . Nếu làm theo kiểu truyền thống thì tất cả các Repository hiện tại và các câu lệnh native query đều phải sửa. Mỗi khi thêm mới function lấy data từ database cũng phải sửa…

@Aspect  
@Component

* Nhược điểm là khó debug và AOP là khái niệm khá trừu tượng

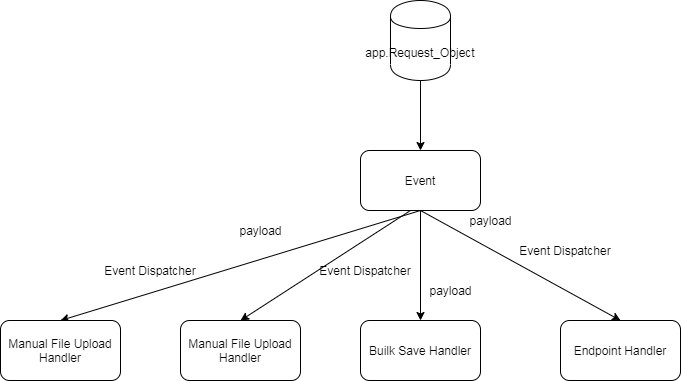
1. **Approval process**

**Requirement & Purpose**

* Phoenix cho phép người dùng tạo request và người dùng khác quyền cao hơn approve request.
* Thiết kế để các request được lưu tập trung, tránh thiết kế lặp lại dữ liệu ở bảng tạm và bảng chính. Dữ liệu tại bảng Request\_Object được lưu ở dạng byte[] để không cần quan tâm đến kiểu dữ liệu.

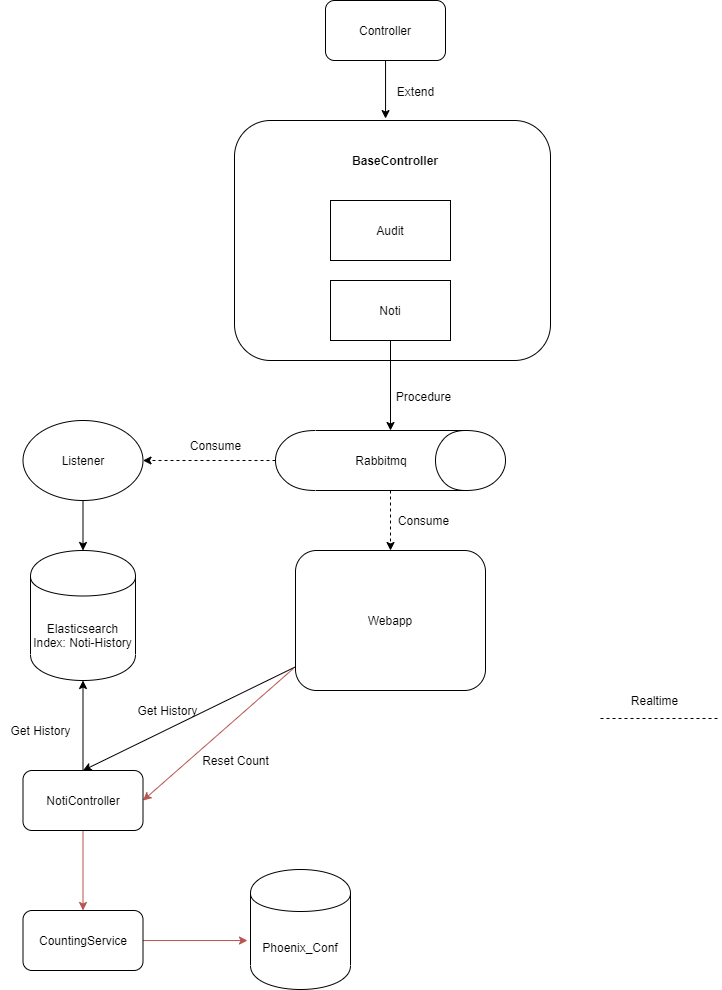
\*\*Note: Có thể cải tiến việc compress byte array bằng thư viện snappy của google. <https://github.com/google/snappy>

* Client cũng cần gọi 1 endpoint duy nhất để tích hợp approval, Sử dụng EDA (event driven architecture) để tách biệt các logic xử lý của các loại request.
  1. DB\_SAVE – cho các table chỉ có CRUD
  2. ENDPOINT - Các pages unfix có logic xử lý khác nhau với mỗi pages. Mỗi phần có 1 internal service riêng. Sử dụng Reflection rất khó để gọi service với input phức tạp nên đã gọi thông qua endpoint (có sẵn)
  3. MANUAL\_File - Upload manual file
  4. ASSUMPTION\_FILE - Import cho cashflow assumption\
* Sử dụng EDA để coi các request có kiểu là các event tương ứng. Với mỗi event sẽ có một handler riêng, kèm theo input data (payload) của mỗi loại sẽ khác nhau



*\*\* Note: Hiện tại các request để xử lý data từ file thì webapp sẽ gửi thông tin FTP và thông tin của file trên server. Webapp tự upload lên thư mục temporary trên server, sau đó mới gửi request sang phoenix-api. API đọc thông tin của file rồi kết nối lên server để lấy file về xử lý.*

1. **Notification**
   * + 1. **Flow Chart**



* Các controller của parameters config, unfix, archiving... đều extend từ BaseController
* BaseController có hàm sendToNoti để routing các loại action trên thành các message tương ứng. Các controller đều phải truyền thông tin của object đang thao tác xuống sendToNoti. Hàm này sử dụng thông tin của object để tạo ra message tương ứng. VD: {table\_name} is changed by; {job\_name} is completed by {user} at {time}
* Hàm sendToNoti gửi message vào rabbitMQ (producer) . Có 2 consumer là webapp để nhận message realtime hiển thị lên push notification và 1 service trong phoenix-api listen để lưu dữ liệu lịch sử
* Mặc định trong phoenix có bind queue Noti với exchange Noti để, Service RabbitListener khi có dữ liệu về queue Noti sẽ transform và thành history data của từng user và đẩy dữ liệu vào elasticsearch làm dữ liệu lịch sử
* Với mỗi lượt user xem history sẽ cần reset số đếm trên giao diện. Việc này được thực hiện bởi counting service. Dữ liệu được lưu trong database Phoenix\_Conf.