FULL TEXT SEARCH USING HIBERNATE SEARCH

BY: AJAY JAGANNATH (1636203)

Table Of Contents

Full-Text Search on database entities using hibernate search ORM + elastic search with Quarkus.

DESCRIPTION

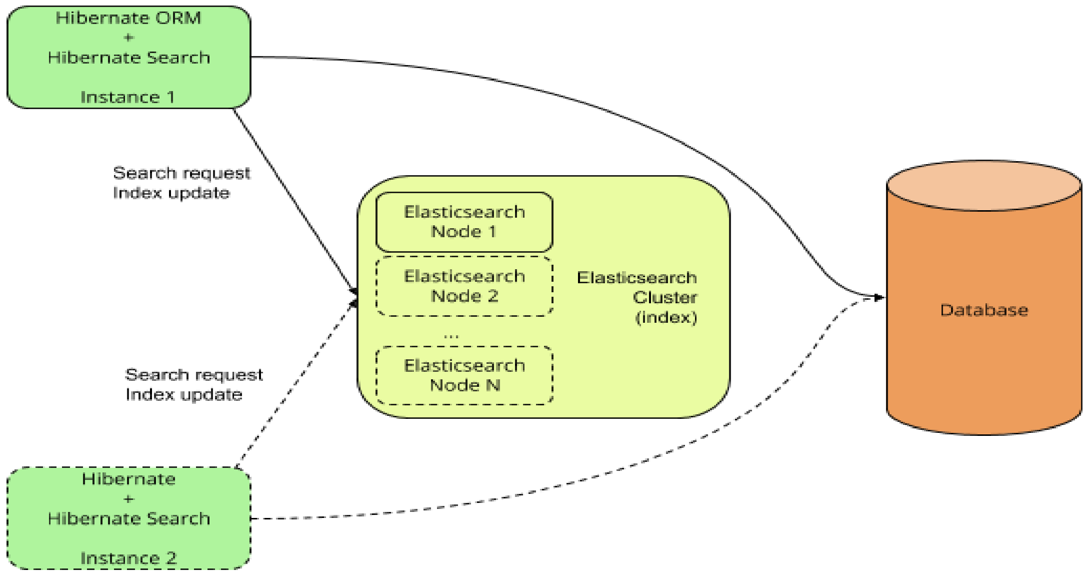
a) Quarkus program with Maven structure importing the following extensions:
   - Hibernate ORM with Panache,
   - the PostgreSQL JDBC driver,
   - Hibernate Search + Elasticsearch,
   - RESTEasy and Jackson.
   - SmallRye OpenAPI

b) Created classes/entity Author and Book which are OneToMany/ManyToOne connected to each other.

c) They act as tables connected to PostgreSQL database where instances of entity can be accessed and altered using REST API annotations.

d) Elasticsearch is used in tandem with REST API functions to perform Full-Text search on the entities using various analyzers and normalizers.

e) The data can be accessed at localhost:8080 or postman when the program is up and running or for a user-friendly version Swagger-UI.



Hibernate ORM + Hibernate Search Instance 1
Search request Index update
Elasticsearch Node 1
Elasticsearch Node 2
...
Elasticsearch Node N
Elasticsearch Cluster (index)
Database
Search request Index update
Hibernate + Hibernate Search Instance 2

Tech Stack:

I. jdk-11.0.11
II. Maven 3.6.3
III. Quarkus 1.13.7
IV. IntelliJ IDEA Community Edition 2021.1.2
V. Elasticsearch-7.13.2-windows-x86_64
VI. Postman 8.6.2
VII. pgAdmin 4 v5

Hibernate Search

Definition

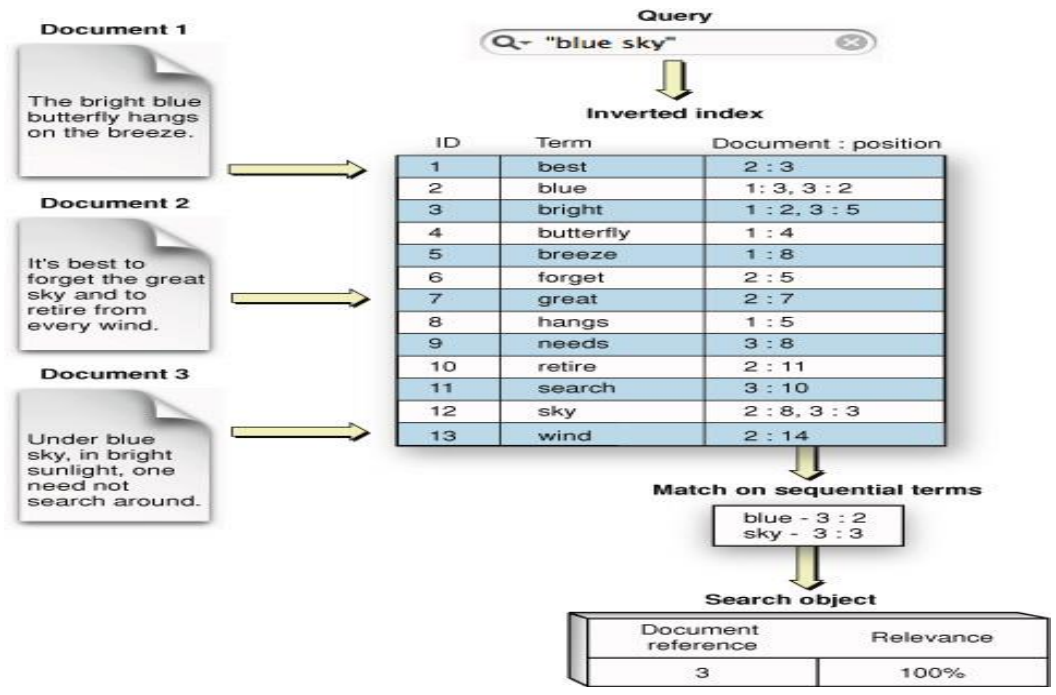
Hibernate Search integrates Elastic Search/Apache Lucene with Hibernate ORM to create a Full Text search Engine in java.

Hibernate Search vs DBMS

When: Full-text search systems excel at quickly and efficiently searching large volumes of text with mostly an abundance of text.

Indexing: Full-text search systems rely on an index in order to perform queries. When the number of documents to search is potentially large, or the quantity of search queries to perform is substantial, the problem of full-text search is often divided into two tasks: indexing and searching. The indexing stage will scan the text of all the documents and build a list of search terms. In the search stage, when performing a specific query, only the index is referenced, rather than the text of the original documents. Full-text search engines also have relevancy ranking capabilities to determine the best match for a query.

Inverted Indexing: The most common type of index is an inverted index, which counts every term, every word, number, etc. in every document with an indication of which documents contain that term and where they occur in the documents. This can result in multi-fold speed increase in performance.


Document 1
The bright blue butterfly hangs on the breeze.
Document 2
It's best to forget the great sky and to retire from every wind.
Document 3
Under blue sky, in bright sunlight, one need not search around.
Query
"blue sky"
Inverted index
Match on sequential terms
blue - 3 : 2
sky - 3 : 3
Search object

ID	Term	Document : position
1	best	2 : 3
2	blue	1 : 3, 3 : 2
3	bright	1 : 2, 3 : 5
4	butterfly	1 : 4
5	breeze	1 : 8
6	forget	2 : 5
7	great	2 : 7
8	hangs	1 : 5
9	needs	3 : 8
10	retire	2 : 11
11	search	3 : 10
12	sky	2 : 8, 3 : 3
13	wind	2 : 14

Document reference	Relevance
3	100%

Disadvantages of Hibernate Search

True Synonyms: Synonyms are two words that mean the same thing in one language. In full-text searching, synonyms hinder effective information retrieval where the system only returns results that match the term and does not return results that refer its synonyms. Ex. Fight and Brawl have similar meanings but searching for one will not show results for the other.

Variant Spellings: Words that mean the exact same thing can sometimes be spelled differently, as in variant British and American spellings. In full-text searching, a search for "Colour" will miss results that use the spelling "color". Fuzzy Search can to an extent negate this problem.

Shortened Forms of Terms: Abbreviations, acronyms, and initialisms can hinder full-text search because a document may contain only the short form of the word or only the long form.

The Homonym Problem: The homonym problem occurs in full-text searching when a single word or phrase has more than one meaning. For example, “Cricket” can mean either the insect or the sport. Both results will show up, with no regard to the context.

Tabulating the Speed of Hibernate Search

Performed for a database with 210 records

Avg time calculated for 10 query executions in m.sec

QUERY USED

Hibernate: /library/book/search?pattern=the

SQL: SELECT * FROM book WHERE LOWER(title) LIKE '%the%'

TABULATION

Hibernate Search (time in m.sec)	SQL with Indexing (time in m.sec)	SQL without indexing (time in m.sec)
18 16 9 13 15 10 16 7 9 11	52 54 64 80 83 67 52 66 71 50	108 131 93 78 97 75 90 113 91 93
Avg Time	Avg Time	Avg Time
12.4	63.9	96.9

Searching for all record in class Book where the title has 'the' in the title.

The time taken to pre-index the records is NOT considered in hibernate search.

INFERENCE

1. Hibernate Search is substantially faster than SQL.
2. Hibernate Search has records ordered by their 'score'.
3. Indexing of table in SQL decreases average time reasonably.
4. The effect of indexing may be more substantial for a larger database.
5. The fluctuation in time can be accounted to the number of records being small, meaning the time scale is small and the effects of random errors is significant.
6. Hence, we can say that the times found here only give the rough idea of the speed.

HIBERNATE SEARCH ANNOTATIONS

A) In the Entity Class

Package: org.hibernate.search.mapper.pojo.mapping.definition.annotation

@Indexed	Subclasses that inherit the @Indexed annotation and will also be indexed by default. Each indexed subclass will have its own index, but they can be seen as being unique to each other as well.
@IndexedEmbedded	It instructs Hibernate Search to *embed* the fields of an associated object into the main object. In the example below, it will instruct Hibernate Search to embed object books defined in list<Book> into Author, creating the field author.books.
@FullTextField	A text field whose value is considered as multiple words. Only works for String fields. Full-text fields should be assigned an analyzer, referenced by its name.
@KeywordField	A text field whose value is considered as a single keyword. Only works for String fields. Keyword fields only contain one token. Keyword fields may be assigned a normalizer.

B) Field annotation attributes

Sortable	Whether the field can be sorted on, i.e., whether a specific data structure is added to the index to allow efficient sorts when querying. Values: Sortable.YES, Sortable.NO, Sortable.DEFAULT.
Name	The name of the index field. By default, it is the same as the property name.
Analyzer	The analyzer used to apply to field values when indexing and querying. Only available on @FullTextField. By default, the analyzer named default will be used.
Normalizer	The normalizer used to apply to field values when indexing and querying. Only available on @KeywordField.

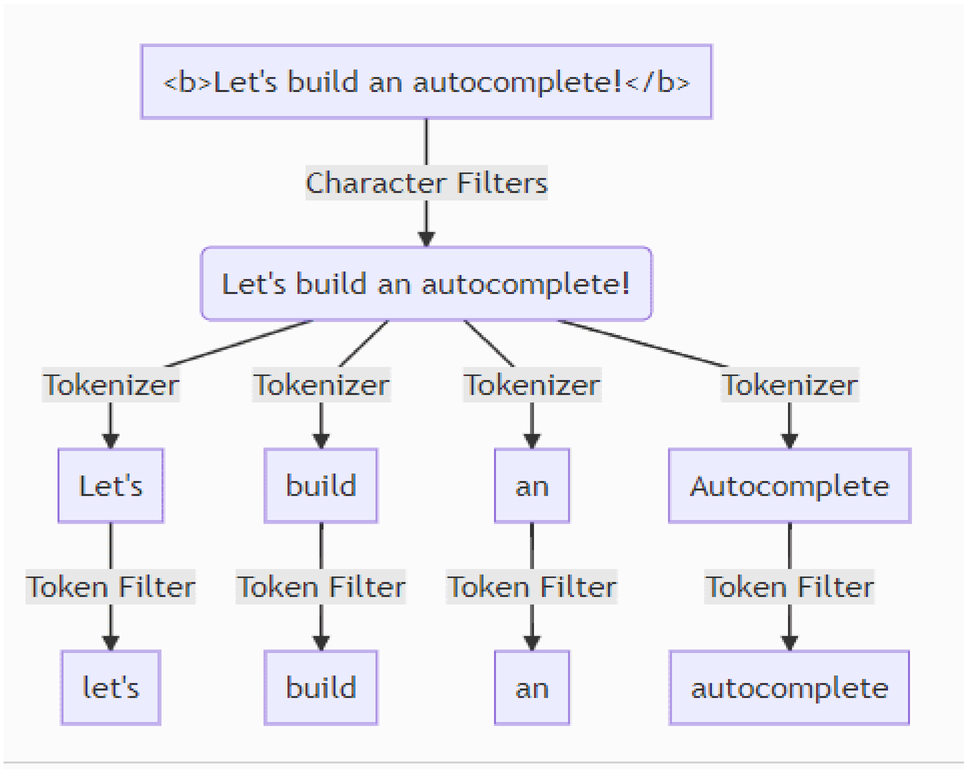
C) Packages used in Resource file

Package	Description
org.hibernate.search.backend.elasticsearch.analysis.ElasticsearchAnalysisConfigurer;	An object responsible for configuring analysis in an Elasticsearch index, providing analysis-related definitions that can be referenced from the mapping.
org.hibernate.search.backend.elasticsearch.analysis.ElasticsearchAnalysisConfigurationContext;	A context allowing the definition of named analysis-related elements in an Elasticsearch backend.
org.hibernate.search.mapper.orm.session.SearchSession;	Initiate the building of a search query. The query will target the indexes mapped to the given type, or to any of its sub-types. MassIndexer: Creates a MassIndexer to rebuild the indexes of all indexed entity types.

ANALYZERS

DEFINITION

Analyzers pre-process input-text submitted for Full Text Search; typically, by removing characters that might prohibit certain match-options. Analysis is performed on document-contents when indexes are created; and is also performed on the input-text submitted for a search. The benefit of analysis is often referred to as language awareness.



NORMALIZER

Normalizer is used to perform Text normalization, which is the process of transforming text into a single canonical form that it might not have had before. Normalizing text before storing or processing it allows for separation of concerns, since input is guaranteed to be consistent before operations are performed on it.

ElasticsearchAnalysisConfigurer

Earlier, we can define the analyzer directly in the entity class using Hibernate Search Annotations.

This feature has now been depreciated in favour of the ElasticsearchAnalysisConfigurer.

We can define the analyzers, normalizers, tokenizers, charfilters, tokenFilters used in our entity class here using the class ElasticsearchAnalysisConfigurerContext.

We can define an pre-existing Analyzer type or create a custom analyzer by defining the various tokenizers, tokenFilter and charFilters and defining Parameters.

Tokenizer: How to Split a string into words/tokens. Works only for analyzer.

TokenFilter: How to alter tokens/words to simplify search criteria. Works for both analyzer and normalizer.

ANALYZERS USED IN THIS PROJECT

A) Name

Tokenizer: Standard

Token Filter: asciifolding, lowercase, edge_Ngram_filter

Edge_Ngram_filter: min =3 letters; max=6 letters

Edge Ngram filter converts created new tokens from existing tokens from their prefix. The length of the prefix goes from the minimum to maximum value defined. Since so many tokens are created, it is a slow process in case the database is large.

Used for: Author firstName and lastName.

B) English

Tokenizer: Standard

Token Filter: asciifolding, lowercase, porter-stem

Porter-stem: Used to convert English words into their -root form. This helps in taking into account the same word existing in different forms.

EX. Mystery and Mysterious have the same root word and searching for either will yield both.

Used for: Book Title

SCORE OF TEXT SEARCH

Text search assigns a score to each document that contains the search term in the indexed fields. The score is an indicator of an item's relevance in the context of the current query. The higher the score, the more relevant the item. The scores can also be manually boosted to increase the priority of certain items depending on requirements.

$$TF * \log(N/df)$$

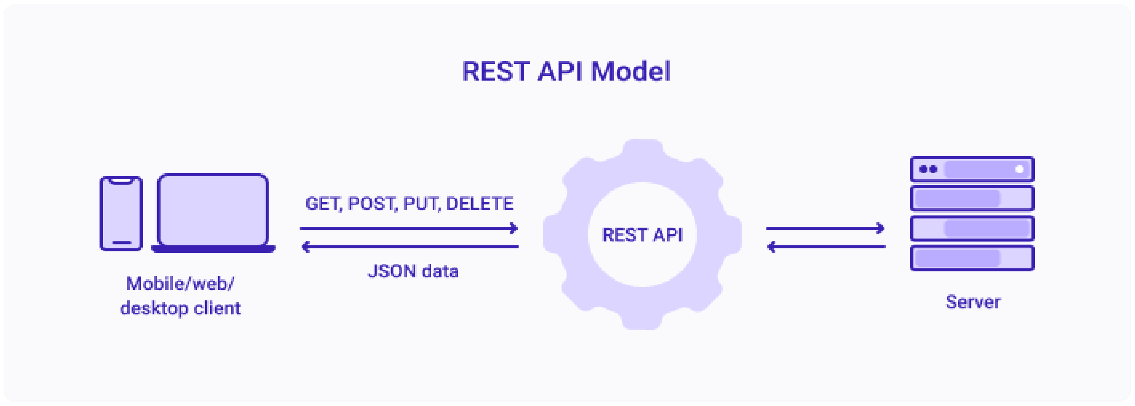
TF = Number of occurrences of word in document
N = Total number of document in corpus
df = Number of documents containing the word

REST API

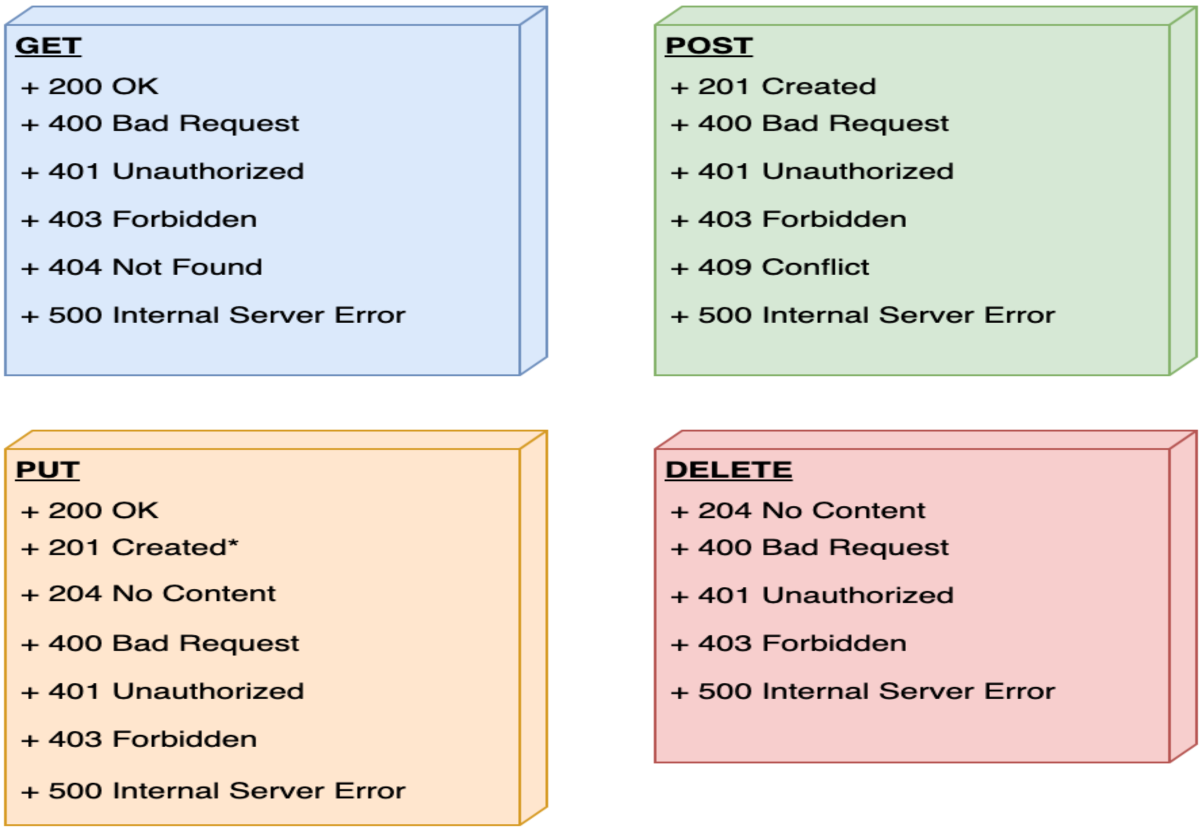
DEFINITION

REST determines how the API (API is an application programming interface. It is a set of rules that allow programs to talk to each other.) looks like. It stands for “Representational State Transfer”. It is a set of rules that developers follow when they create their API. It acts as an interface between the user(client) and the developer(server) to facilitate the communication between them.

Each URL is called a request while the data sent back to you is called a response.



GET	This request is used to get a resource from a server. If we perform a `GET` request, the server looks for the data you requested and sends it back to you. In other words, a `GET` is analogous to `READ` operation. This is the default request method
POST	This request is used to create a new resource on a server. If you perform a `POST` request, the server creates a new entry in the database and tells you whether the creation is successful. In other words, a `POST` is analogous to `CREATE` operation.
PUT	This request is used to update a resource on a server. If you perform a `PUT` request, the server updates an entry in the database and tells you whether the update is successful. In other words, a `PUT` is analogous to `UPDATE` operation.
DELETE	This request is used to delete a resource from a server. If you perform a `DELETE` request, the server deletes an entry in the database and tells you whether the deletion is successful.



GET
+ 200 OK
+ 400 Bad Request
+ 401 Unauthorized
+ 403 Forbidden
+ 404 Not Found
+ 500 Internal Server Error

POST
+ 201 Created
+ 400 Bad Request
+ 401 Unauthorized
+ 403 Forbidden
+ 409 Conflict
+ 500 Internal Server Error

PUT
+ 200 OK
+ 201 Created*
+ 204 No Content
+ 400 Bad Request
+ 401 Unauthorized
+ 403 Forbidden
+ 500 Internal Server Error

DELETE
+ 204 No Content
+ 400 Bad Request
+ 401 Unauthorized
+ 403 Forbidden
+ 500 Internal Server Error

REST API METHODS USED

A) FOR CLASS AUTHOR

1) @POST

Path	/library/author
Input	2 Words- Author firstName and lastName
Description	Add a new instance to class Author

2) @PUT

Path	/library/author/{id}
Input	An integer and 2 words- Author ID, firstName and lastName
Description	If the author exists, then update their data. Else no task to be done.

3) @DELETE

Path	/library/author/{id}
Input	An integer-Author ID
Description	If the author exists, then delete their data. Else no task to be done. The books written by author will automatically get deleted.

4) @GET

Path	/library/authors
Input	nil
Description	All instances in class Author are returned.
Sort	First by lastName, then by firstName

Path	/library/authors/time
Input	nil
Description	The time taken to get instances in class Author are returned. (Note: time in sec)
Sort	First by lastName, then by firstName

Path	/library/author/{id}
Input	An integer- Author ID
Description	If the author exists, then their info is returned

Path	/library/author/search
Input	A line- Author firstName or lastName or one of their Books Title (full/partial)
Description	To search and return instances(list) of Class Author if the input matches one of its three fields based on analyzer definition.
Sort	First by lastName, then by firstName

Path	/library/author/time/search
Input	A line- Author firstName or lastName or one of their Books Title (full/partial)
Description	Time taken to search and return instances(list) of Class Author if the input matches one of its three fields
Sort	First by lastName, then by firstName

B) FOR CLASS BOOK

1) @POST

Path	/library/book
Input	A Line and an integer- Book Title and Author Id
Description	If the author exists, add a new instance to class Book written by them. Else no task to be done.

2) @DELETE

Path	/library/book/{id}
Input	An integer-Book ID
Description	If the author exists, then delete their data. Else no task to be done.

3) @GET

Path	/library/books
Input	nil
Description	All instances in class Book are returned.
Sort	Book title

Path	/library/books/time
Input	nil
Description	The time taken to get instances in class Book are returned. (Note: time in sec)
Sort	Book Title

Path	/library/books/page
Input	2 integers, page no and no of books per page
Description	Return all Instances of book in a certain page
Sort	Book Title

Path	/library/book/{id}
Input	An integer- Book ID
Description	If the book exists, then its info is returned

Path	/library/book/search
Input	A line- Book Title (full/partial)
Description	To search and return instances(list) of Class Book if the input matches one of its fields based on analyzer definition.
Sort	Score

Path	/library/book/time/search
Input	A line- Book Title (full/partial)
Description	Time taken to search instances(list) of Class Book if the input matches one of its fields based on analyzer definition.
Sort	Score

Path	/library/book/wildcard/search
Input	A line- Book Title (full/partial)
Description	A wildcard is a character that can be used to substitute for another character or a set of characters. * implies zero or more characters, while ? implies one character. To search and return instances(list) of Class Book if the input matches one of its fields based on analyzer definition allowing wildcard characters.
Sort	Score

Path	/library/book/phrase/slop/search
Input	A Phrase- Book Title (full/partial)
Description	Using Phrase search means that the exact phrase given by the user must be contained in the field being searched in the same order and no spacing. Slop can make an exception to this where; we can set a limit on the number of words that can appear between the different words of the phrases. To search and return instances(list) of Class Book if the input matches one of its fields based on analyzer definition such that the phrase is matched allowed a maximum of 2 words between them.
Sort	Score

Path	/library/book/except/search
Input	A line
Description	To search and return instances(list) of Class Book if each word given in the input does NOT appear in the field being searched. I.e., We can define the words that must not appear in any of the book titles
Sort	Score

Path	/library/book/fuzzy/search
Input	A word- Book Title (full/partial)
Description	A fuzzy search searches for text that matches a term closely instead of exactly. Fuzzy searches find relevant results even when the search terms are misspelled. We can define the number of misspelled characters and fix a rigid prefix with no mistakes To search and return instances(list) of Class Book if the input matches one of its fields with a fuzzy match allowing 1 misspelled character appearing NOT before the first 3 characters of the word.
Sort	Score

APPENDIX

RESTEASY Annotations

Difference between @PUT and @POST Rest API annotations

@PUT request is also idempotent in HTTP, which means it will produce the same results if executed once more multiple times.

1. **@GET, @PUT, @POST, @DELETE** are used to access/alter the instances of entity located by the @Path annotation.

2. **@Consumes** annotation is used to specify which MIME media types of representations a resource can accept, or consume, from the client. If @Consumes is applied at the class level, all the response methods accept the specified MIME types by default.
   If @Consumes is applied at the method level, it overrides any @Consumes annotations applied at the class level.

3. **@Transactional**: At a high level, Javax creates proxies for all the classes annotated with *@Transactional*, either on the class or on any of the methods. The proxy allows the framework to inject transactional logic before and after the running method, mainly for starting and committing the transaction. We have to note that only public method can be annotated with it and also proxies are created only for external calls.

RESTEASY JAX-RS Annotations

1. **@FormParam**: The annotation @FormParam can be used to inject the parameters of a Web form into a RESTful Web service.

2. **@PathParam:** The @PathParam annotation binds the value of a path segment to a resource method parameter. Ex. @Path("/author/{id}") then this id can be binded to a parameter.

3. **@QueryParam:** the @QueryParam annotation binds the value of path segment to a resource method parameter. Ex. Localhost:8080/library/author/search?pattern="abc", now will be mapped to the parameter.

Java Persistence Annotations

1. **@Entity**: Specifies that the class is an entity. This annotation is applied to the entity class, implying that the database must identify this class as a table.

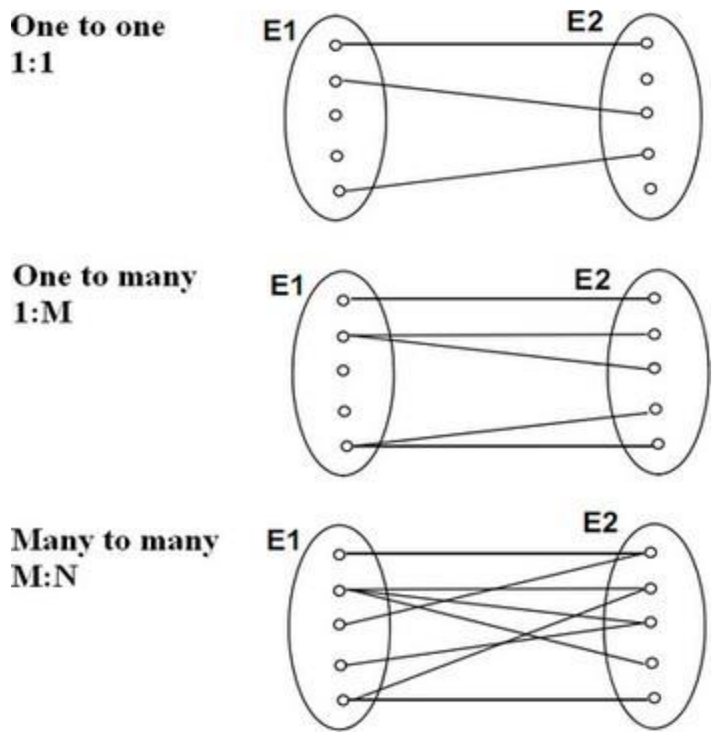
2. **@Table**: Specifies the primary table for the annotated entity. If no Table annotation is specified for an entity class, the default values apply.

3. **@Column**: Specifies the mapped column for a persistent property or field. If no Column annotation is specified, the default values apply.

4. **@OneToMany**: Specifies a many-valued association with one-to-many multiplicity. If the collection is defined using generics to specify the element type, the associated target entity type need not be specified; otherwise, the target entity class must be specified.

5. **@ManyToOne**: The @ManyToOne annotation may be used within an embeddable class to specify a relationship from the embeddable class to an entity class.

Using the OneToMany and ManyToOne together ensures bidirectional relationship between the embedded class and the entity class.



Mapping Parameters

1. **CascadeType**: Defines the set of cascadable operations that are propagated to the associated entity. The value cascade=ALL is equivalent to cascade {PERSIST, MERGE, REMOVE, REFRESH, DETACH}.

2. **OrphanRemoval**: Used to remove Child class instance which is not associated with any Parent class entity. Hence, when we delete an author class instance, we don't have to delete the book instances under it, as it will get deleted automatically.

3. **FetchType**: There are two ways to fetch child class instance when the parent class instance is fetched.
    a. EAGER: To load it together with the rest of the fields
    b. LAZY: To load it on-demand

When a parent class instance has many child instances it is not efficient to load all of it together with it, especially when they are not needed and, in that case, we can declare that we want child class instances to be loaded when they are actually needed. This is called lazy loading.

@JsonIgnore: It is used to avoid infinite loops when serializing with Jackson

@Inject: Identifies injectable constructors, methods, and fields. Constructors are injected first, followed by fields, and then methods. Fields and methods in superclasses are injected before those in subclasses. Ordering of injection among fields and among methods in the same class is not specified.

Open API Annotations

@Operation: Used to Describe a single API operation on a path to the end-user. The summary, operation Id, description of the Rest API operation can be defined using this annotation.

@Parameter: Used to Describe a single operation parameter to the end-user. The Description, example, schema and if the field is a required one can be defined using this annotation.

Other annotations such as the @APIResponse which Describes a single response from an API operation post its completion have not been used in this project.

RESOURCES

I. https://tefkos.comminfo.rutgers.edu/Courses/e530/Readings/Beal%202008%20full%20text%20searching.pdf

II. https://quarkus.io/guides/hibernate-search-orm-elasticsearch

III. https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html

IV. https://www.baeldung.com/hibernate-search

V. https://docs.jboss.org/hibernate/stable/search/reference/en-US/html_single/

VI. https://thorben-janssen.com/add-full-text-search-application-hibernate-search/

VII. https://lucidworks.com/post/full-text-search-engines-vs-dbms/

VIII. https://openliberty.io/guides/microprofile-openapi.html#augmenting-the-existing-jax-rs-annotations-with-openapi-annotations

IX. https://swagger.io/tools/swaggerhub/

X. https://mvnrepository.com/artifact/io.quarkus