Spring Boot and MongoDB

For swagger, we have to get 2 dependencies: swagger ui (spring fox) and swagger 2 (spring fox).
While making spring boot application add dependencies like spring web, spring data mongo.

For mongoDB setup:
We can put properties in application.properties:

spring.data.mongodb.uri=
spring.data.mongodb.database=

https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api

https://document360.com/blog/swagger-api/

https://www.geeksforgeeks.org/spring-boot-rest-api-documentation-using-swagger/

To enable the swagger for a controller:

```
@RequestMapping("/")
public void redirect(HttpServletResponse response) throws IOException {
	response.redirect("/swagger-ui.html");
}
```

To remove predefined request url by swagger we use @ApiIgnore

```
@RequestMapping("/")
@ApiIgnore
public void redirect(HttpServletResponse response) throws IOException {
	response.redirect("/swagger-ui.html");
}
```

For repository, we have to extend our custom interface with MongoRepository<Post, String> here post is entity and String means primary key is of type string.

To map the entity with the collection (We say table in sql), we use
@Document(collection="JobPost")
Is just the same as @Entity(table="JobPost")

Atlas Search

To have an effective search system where we want to search something based

on like say text included in the fields we can use index.
Go to cloud where we browse collection there we can find above collection -> search index

We can create using visual editor by default it will pick by dynamic field that is all field but we can make it like static by including some fields as needed.

Go to aggregation section where we can pipeline the activities like first searching, then sorting, then limiting.

Go to add stage and define the pipelined operations one by one.

In the index section of the search stage we can mention the index we are using.

```
{
	index : <name_of_index>
	text : {
		query : ‘java’
		path : [name of fields, if there is one we can directly write it as string]
	}
}
```

In sort stage:

```
{
	exp : 1
}
```

This sorts the document on the basis of experience in the ascending order. -1 for descending

In the limit stage:
Simply mention the number (as digit) how many documents we want from the search.

Client Side field encryption

Field will be stored as encryption and decrypted when accessing them.

Now to get the search thing we made, you can find the export to language button and specify the language java to get the code.

Make a new SearchRepositoryImpl class and put the logic there.

Be aware you have to auto-wire 2 classes i.e. MongoClient, MongoConverter.
MongoClient helps to establish connection with the database at cloud.
MongoConverter helps to convert the document to entity.

AggregateIterable<Document> result = collection.aggregate(here the search logic will be there);

result.forEach(doc -> posts.add(converter.read(Post.class, doc));

To solve cross-origin problem (to allow other server/client on other machine or same machine to access our data)

At top of controller use @CrossOrigin(origins="http://localhost:3000")
Over all the methods we can just put @CrossOrigin