PAGINATION

* Basically this is a technique used to get the large amount of data into small chunks so that i can be easily accessible as well as understandable.
* Two fileds it inculdes : Page Number and Page Size
* Pageable Class is used to set the pageNumber and pageSize (Take dynamiclly using @RequestParam)
  + PageRequest.of(pageNumber, pageSize) : It will return the object of Pageable Class
  + Then we have to use the Page<T> class to find all the pages (data) using repo.findAll('objectOfPagable')
  + Get data using List<T> or something according to the requirements and use '.getContent()' method on the Page Class's object created on 2nd Step.
  + NOTE\*\* In URL or the path the pagingNumber starts from 0

---------------------------------------------------------------------------------------------------------------------------------------

CriteriaBuilder

CriteriaBuilder in Spring Boot, part of the JPA (Java Persistence API), is used for dynamically constructing queries to interact with databases.

It provides a programmatic way to build queries directly in Java code, making it flexible and adaptable to different search criteria.

Here are some useful details and concepts commonly used with CriteriaBuilder:

**I) EntityManager and CriteriaBuilder Initialization:**

* In Spring Boot, CriteriaBuilder is typically obtained from the EntityManager:

@PersistenceContext

* private EntityManager entityManager;
* CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder();

**II) Creating CriteriaQuery:**

* CriteriaQuery is used to define the structure of the query.
* It represents the root entity and any additional query elements (select, where, order by, etc.):
* CriteriaQuery<EntityType> criteriaQuery = criteriaBuilder.createQuery(EntityType.class);

**III) Root and Path:**

* Root<EntityType> represents the entity being queried
* Root<EntityType> root = criteriaQuery.from(EntityType.class); --> This line specifies

(SELECT \* FROM EntityType)

**IV) Predicates:**

* Predicates are conditions added to the query using CriteriaBuilder to filter the results. These can be combined with and, or, etc.:
* Predicate condition = criteriaBuilder.equal(root.get("fieldName"), value);
* criteriaQuery.where(condition);

**V) Sorting:**

* CriteriaBuilder allows specifying sorting using orderBy
* criteriaQuery.orderBy(criteriaBuilder.asc(root.get("fieldName")));

**VI) Aggregation Functions:**

* CriteriaBuilder supports aggregation functions such as count, sum, avg, max, min:
* criteriaQuery.select(criteriaBuilder.count(root));

**VII) Joins:**

* For querying across multiple entities, CriteriaBuilder supports different types of joins (innerJoin, leftJoin, rightJoin):
* Join<EntityType, OtherEntityType> join = root.join("otherEntity", JoinType.INNER);

**VIII) Executing the Query:**

* Once the criteria query is constructed, it can be executed using the EntityManager:
* TypedQuery<EntityType> typedQuery = entityManager.createQuery(criteriaQuery);
* List<EntityType> results = typedQuery.getResultList();

---------------------------------------------------------------------------------------------------------------------------------------

STEP-BY-STEP QUERIES FOR CriteriaBuilder

1. @PersistenceContext

private EntityManager entityManager;

1. CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder();
2. CriteriaQuery<EntityType> criteriaQuery = criteriaBuilder.createQuery(EntityType.class)
3. Root<EntityType> root = criteriaQuery.from(EntityType.class);
4. Predicate condition = criteriaBuilder.equal(root.get("fieldName"), value);
   1. criteriaQuery.where(condition);
   2. (Optional) criteriaQuery.orderBy(criteriaBuilder.asc(root.get("fieldName")));
   3. (Optional) criteriaQuery.select(criteriaBuilder.count(root));
5. TypedQuery<EntityType> typedQuery = entityManager.createQuery(criteriaQuery);
   1. List<EntityType> results = typedQuery.getResultList();

--------------------------------------------------------------------------------------------------------------------------------------------------

Easy Way to Implement the Pagination and Search Using CriteriaBuilder

1. Create PageRequestDto to get the specified data

@Getter  
@Setter  
public class PaginationResponseDto<T> {  
  
 private List<T> content;  
 private int pageNumber;  
 private int pageSize;  
 private long totalElements;  
 private int totalPages;  
  
 public PaginationResponseDto(List<T> content, int pageNumber, int pageSize, long totalElements, int totalPages) {  
 this.content = content;  
 this.pageNumber = pageNumber;  
 this.pageSize = pageSize;  
 this.totalElements = totalElements;  
 this.totalPages = totalPages;  
 }  
}

1. Make Method in Service to retrieve the data

public PaginationResponseDto<AppUserDto> getAppUserByNameOrEmail(String searchValue, int pageNumber, int pageSize);

1. Provide ServiceImpl

public PaginationResponseDto<AppUserDto> getAppUserByNameOrEmail(String searchValue,

int pageNumber, int pageSize)

throws ResourceNotFoundException {  
  
 Specification<AppUser> appUserSpecification = new Specification<AppUser>() {  
 @Override  
 public Predicate toPredicate(Root<AppUser> root, CriteriaQuery<?> query, CriteriaBuilder cb) {  
 Predicate namePredicate = cb.like(cb.lower(root.get("username")),

"%" + searchValue.toLowerCase() + "%");  
 Predicate emailPredicate = cb.like(cb.lower(root.get("email")),

"%" + searchValue.toLowerCase() + "%");  
 return cb.or(namePredicate, emailPredicate);  
 }  
 };  
  
 PageRequest pageRequest = PageRequest.*of*(pageNumber, pageSize);  
 Page<AppUser> appUserPage = appUserRepository.findAll (appUserSpecification, pageRequest);  
  
 List<AppUserDto> appUserDtoList = appUserPage.stream().map(AppUserDto::new).toList();  
  
 return new PaginationResponseDto<>(  
 appUserDtoList,  
 appUserPage.getNumber(),  
 appUserPage.getSize(),  
 appUserPage.getTotalElements(),  
 appUserPage.getTotalPages()  
 );  
}

1. Make Controller to get the data from APIs and send it to ServiceImpl

@PostMapping("/pagination/getByNameOrEmail")  
public ResponseEntity<PaginationResponseDto<AppUserDto>> getAppUserByCriteria(@RequestParam(required = false) String searchValue,  
 @RequestParam(defaultValue = "0") int page,  
 @RequestParam(defaultValue = "10") int size){  
 return ResponseEntity.*ok*(appUserService.getAppUserByNameOrEmail(searchValue, page, size));  
}

Now hit API POST request in POSTMAN for /pagination/getByNameOrEmail?searchValue=&page

------------------------------------------------------------------------------------------------------------------------------------