Pagination in Spring JPA – 2024

Pagination can be achieved using the following approaches.

* Using JpaRepository
* Using PagingAndSortingRepository
* Using CrudRepository
* Using **Slice** in CrudRepository

**Difference between JpaRespository, PagingAndSortingRepository and CrudRepository**

CrudRepository mainly provides CRUD functions.

PagingAndSortingRepository provides methods to do pagination and sorting records.

JpaRepository provides some JPA-related methods such as flushing the persistence context and deleting records in a batch.

**public** **interface** **JpaRepository** **extends** **PagingAndSortingRepository**

**public** **interface** **PagingAndSortingRepository** **extends** **CrudRepository**

**public** **interface** **CrudRepository** **extends** **Repository**

**JpaRepository 🡺 PagingAndSortingRepository 🡺 CrudRepository 🡺 Repository**

**Example is given bewlow.**

**Entity Layer**

@Entity(name = "AppUser")

@Table(name = "app\_user")

@Getter

@Setter

@ToString

@NoArgsConstructor

**public** **class** AppUser {

@Id

@GeneratedValue

**private** Long id;

@Column(name = "name")

**private** String name;

@Column(name = "city")

**private** String city;

**public** AppUser(String name, String city) {

**super**();

**this**.name = name;

**this**.city = city;

}

}

Repository Layer

Using JpaRepository

@Repository

**public** **interface** AppUserJPARepository **extends** JpaRepository<AppUser, Long> {

}

Using PagingAndSortingRepository

@Repository

**public** **interface** AppUserPagingRepository **extends** PagingAndSortingRepository<AppUser, Long> {

}

Using CrudRepository

@Repository

**public** **interface** AppUserCrudRepository **extends** CrudRepository<AppUser, Long> {

Page<AppUser> findAll(Pageable pageable);

@Query("select u from AppUser u")

Page<AppUser> getAllAppUsers(Pageable pageable);

@Query("select u from AppUser u")

Slice<AppUser> getAllSlicedUsers(Pageable pageable);

}

Service Layer

**public** **interface** UserService {

**void** saveAllUsers(List<AppUser> users);

Page<AppUser> getPaginatedUsersUsingJPARepo(**int** pageNo, **int** pageSize);

Page<AppUser> getPaginatedUsersUsingPaginAndSortingRepo(**int** pageNo, **int** pageSize);

Page<AppUser> getPaginatedUsersUsingCrudRepo(**int** pageNo, **int** pageSize);

Page<AppUser> getPaginatedUsersUsingCrudRepoCustom(**int** pageNo, **int** pageSize);

Slice<AppUser> getSlicedPaginatedUsersUsingCrudRepo(**int** pageNo, **int** pageSize);

}

**For Sorting**

We can do that by passing the sorting details into our *PageRequest* object itself:

**Pageable** sortedByName = PageRequest.of(0, 3, Sort.by("name"));

**Pageable** sortedByPriceDesc = PageRequest.of(0, 3, Sort.by("price").descending());

**Pageable** sortedByPriceDescNameAsc =

PageRequest.of(0, 5, Sort.by("price").descending().and(Sort.by("name")));

**PageRequest.*of*(pageNo, pageSize);**

Service Implementation Class

@Service

**public** **class** UserServiceImpl **implements** UserService {

@Autowired

**private** AppUserJPARepository userJpaRepo;🡸 Using JpaRepository

@Autowired

**private** AppUserPagingRepository pagingUserRepo; 🡸 Using PagingAndSortingRepository

@Autowired

**private** AppUserCrudRepository crudRepo; 🡸 Using CrudRepository

@Override

**public** **void** saveAllUsers(List<AppUser> users) {

userJpaRepo.saveAll(users);

}

**public** Page<AppUser> getPaginatedUsersUsingJPARepo(**int** pageNo, **int** pageSize) {

Pageable pageable = PageRequest.*of*(pageNo, pageSize);

**return** userJpaRepo.findAll(pageable);

}

**public** Page<AppUser> getPaginatedUsersUsingPaginAndSortingRepo(**int** pageNo, **int** pageSize) {

Pageable pageable = PageRequest.*of*(pageNo, pageSize);

**return** pagingUserRepo.findAll(pageable);

}

**public** Page<AppUser> getPaginatedUsersUsingCrudRepo(**int** pageNo, **int** pageSize) {

Pageable pageable = PageRequest.*of*(pageNo, pageSize);

**return** crudRepo.findAll(pageable);

}

**public** Page<AppUser> getPaginatedUsersUsingCrudRepoCustom(**int** pageNo, **int** pageSize) {

Pageable pageable = PageRequest.*of*(pageNo, pageSize);

**return** crudRepo.getAllAppUsers(pageable);

}

**public** Slice<AppUser> getSlicedPaginatedUsersUsingCrudRepo(**int** pageNo, **int** pageSize) {

Pageable pageable = PageRequest.*of*(pageNo, pageSize);

**return** crudRepo.getAllSlicedUsers(pageable);

}

}

AutoRun Class for Testing

@Component

**public** **class** AutoRun {

@Autowired

**private** UserService userService;

**public** **void** persistAllUsers() {

Instant start = Instant.*now*();

List<AppUser> users = **new** ArrayList<>();

**for** (**int** i = 1001; i < 1500; i++) {

users.add(**new** AppUser("Name-" + i, "City-" + i));

}

userService.saveAllUsers(users);

Instant end = Instant.*now*();

**int** timeInNonSecs = Duration.*between*(start, end).getNano();

System.***out***.println("Time Taken to complete in nanoseconds: " + timeInNonSecs);

}

**public** **void** showJPARepoPaginatedUsers() {

**for** (**int** i = 0; i < 5; i++) {

Page<AppUser> pagedUsers = userService.getPaginatedUsersUsingJPARepo(i, 10);

List<AppUser> appUserList = pagedUsers.getContent();

appUserList.forEach(user -> System.***out***.println("User: " + user));

}

}

**public** **void** showPagingRepoPaginatedUsers() {

**for** (**int** i = 0; i < 5; i++) {

Page<AppUser> pagedUsers = userService.getPaginatedUsersUsingPaginAndSortingRepo(i, 10);

List<AppUser> appUserList = **pagedUsers.getContent();**

appUserList.forEach(user -> System.***out***.println("User: " + user));

}

}

**public** **void** showCrudRepoPaginatedUsers() {

**for** (**int** i = 0; i < 5; i++) {

Page<AppUser> pagedUsers = userService.getPaginatedUsersUsingCrudRepo(i, 10);

List<AppUser> appUserList = pagedUsers.getContent();

appUserList.forEach(user -> System.***out***.println("User: " + user));

}

}

**public** **void** showCrudRepoPaginatedUsersCustom() {

**for** (**int** i = 0; i < 5; i++) {

Page<AppUser> pagedUsers = userService.getPaginatedUsersUsingCrudRepoCustom(i, 10);

List<AppUser> appUserList = pagedUsers.getContent();

appUserList.forEach(user -> System.***out***.println("User: " + user));

}

}

**public** **void** showSlicedCrudRepoPaginatedUsers() {

**for** (**int** i = 0; i < 10; i++) {

Slice<AppUser> pagedUsers = userService.getSlicedPaginatedUsersUsingCrudRepo(i, 10);

List<AppUser> appUserList = pagedUsers.getContent();

appUserList.forEach(user -> System.***out***.println("User: " + user));

}

}

@EventListener(ApplicationReadyEvent.**class**)

**public** **void** run() {

System.***out***.println("Application running ...");

// persistAllUsers();

// showJPARepoPaginatedUsers();

// showPagingRepoPaginatedUsers();

// showCrudRepoPaginatedUsers();

// showCrudRepoPaginatedUsersCustom();

showSlicedCrudRepoPaginatedUsers();

System.***out***.println("Database operations completed ...");

}

}

Application.properties

server.port=9090

## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)

spring.datasource.url=jdbc:postgresql://localhost:5432/postgres

spring.datasource.username=postgres

spring.datasource.password=postgres

# The SQL dialect makes Hibernate generate better SQL for the chosen database

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect

# Hibernate ddl auto (create, create-drop, validate, update)

spring.jpa.hibernate.ddl-auto = update

spring.jpa.properties.hibernate.generate\_statistics=true

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

spring.jpa.properties.hibernate.temp.use\_jdbc\_metadata\_defaults=false

**public** List<AppUser> getPaginatedUsersUsingCrudRepoCustom(**int** pageNo, **int** pageSize) {

Pageable pageable = PageRequest.*of*(pageNo, pageSize);

**Page<User> pagedUsers =** crudRepo.getAllAppUsers(pageable);

List<User> userList = pagedUser.getContent();

return userList;

}

**Difference between Slice and Page**

Page Extends Slice (Page 🡺Slice)

Page knows total no of elements and pages available

Slice only knows whether there is a next slice available (Useful for large result set)