[Thực hành] Quản lý khách hàng: Phân trang và tìm kiếm

Mục tiêu

Luyện tập triển khai phân trang (pagination) trong Spring Data JPA.

Mô tả

Trong phần này, chúng ta sẽ bổ sung các chức năng sau cho ứng dụng Quản lý khách hàng:

* Phân trang danh sách khách hàng
* Tìm kiếm khách hàng

Hướng dẫn

* Trong bài thực hành này chúng ta sẽ nâng cấp bài thực hành 1. Các bạn có thể tải từ link git cũ của các bạn. Hoặc dùng link git tham khảo sau:

<https://github.com/nguyendongminhtri/MD1-Bai7-TH1-Province-Customer.git>

* Cập nhật **ICustomerRepository**: để trả về Page<Customer> thay vì Iterable<Customer>

package rikkei.academy.repository;  
  
import org.springframework.data.domain.Page;  
import org.springframework.data.domain.Pageable;  
import org.springframework.data.repository.PagingAndSortingRepository;  
import org.springframework.stereotype.Repository;  
import rikkei.academy.model.Customer;  
import rikkei.academy.model.Province;  
  
@Repository  
public interface ICustomerRepository extends PagingAndSortingRepository<Customer, Long> {  
// Iterable<Customer> findAllByProvince(Province province);  
 Page<Customer> findAllByFirstNameContaining(String firstname, Pageable pageable);  
}

* Cập nhật phương thức findAll() của interface **ICustomerService** và **CustomerService** để trả về Page<Customer> thay vì Iterable<Customer>.

Trong đó interface ICustomerService sẽ được cập nhật thành như sau:

package rikkei.academy.service.customer;  
  
import org.springframework.data.domain.Page;  
import org.springframework.data.domain.Pageable;  
import rikkei.academy.model.Customer;  
import rikkei.academy.model.Province;  
import rikkei.academy.service.IGenerateService;  
  
public interface ICustomerService extends IGenerateService<Customer> {  
// Iterable<Customer> findAllByProvince(Province province);  
 Page<Customer> findAll(Pageable pageable);  
 Page<Customer> findAllByFirstNameContaining(String firstname, Pageable pageable);  
  
}

* Trong đó Pageable là 1 đối tượng có chứa 3 thuộc tính page, size và sort hỗ trợ cho việc phân trang. Pageable được tự động ánh xạ từ request. **Lưu ý vì lớp Pageable có ở 2 thư viện để import nên chúng ta sẽ import lớp Pageabe từ thư viện org.springframework.data.domain.Pageable**.
* Page<> là đối tượng chứa danh sách các thực thể và các thuộc tính của một trang. Page<> được trả về trong response.
* Phương thức **findAllByFirstNameContaining** sẽ được Spring Data Repository tự động tạo proxy để thực hiện câu lệnh truy vấn. Thực hiện chức năng tìm kiếm Customer theo **firstname**

package rikkei.academy.service.customer;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.data.domain.Page;  
import org.springframework.data.domain.Pageable;  
import rikkei.academy.model.Customer;  
import rikkei.academy.repository.ICustomerRepository;  
  
import java.util.Optional;  
  
public class CustomerServiceIMPL implements ICustomerService{  
 @Autowired  
 private ICustomerRepository customerRepository;  
  
 @Override  
 public Iterable<Customer> findAll() {  
 return customerRepository.findAll();  
 }  
  
 @Override  
 public Optional<Customer> findById(Long id) {  
 return customerRepository.findById(id);  
 }  
  
 @Override  
 public void save(Customer customer) {  
 customerRepository.save(customer);  
 }  
  
 @Override  
 public void remove(Long id) {  
 customerRepository.deleteById(id);  
 }  
  
// @Override  
// public Iterable<Customer> findAllByProvince(Province province) {  
// return customerRepository.findAllByProvince(province);  
// }  
@Override  
public Page<Customer> findAll(Pageable pageable) {  
 return customerRepository.findAll(pageable);  
}  
  
 @Override  
 public Page<Customer> findAllByFirstNameContaining(String firstname, Pageable pageable) {  
 return customerRepository.findAllByFirstNameContaining(firstname, pageable);  
 }  
}

* Cập nhật **ProvinceController** : Thêm Pageable pageable và sửa trả về Page

@GetMapping("/view-province/{id}")  
 public ModelAndView viewProvince(@PathVariable("id") Long id, Pageable pageable){  
 Optional<Province> provinceOptional = provinceService.findById(id);  
 if(!provinceOptional.isPresent()){  
 return new ModelAndView("/error.404");  
 }  
  
// Iterable<Customer> customers = customerService.findAllByProvince(provinceOptional.get());  
 Page<Customer> customers = customerService.findAll(pageable);  
  
 ModelAndView modelAndView = new ModelAndView("/province/view");  
 modelAndView.addObject("province", provinceOptional.get());  
 modelAndView.addObject("customers", customers);  
 return modelAndView;  
 }

* Cập nhật **CustomerController** sử dụng phương thức findAll() mới của **CustomerService**

Phương thức listCustomers() có thêm tham số Pageable, là đối tượng được mapping tự động từ URL. Pageable được tạo ra từ 3 tham số là page, size và sort. Ví dụ: /customers?page=1&size=10&sort=firstname.

package rikkei.academy.controller;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.data.domain.Page;  
import org.springframework.data.domain.Pageable;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.web.servlet.ModelAndView;  
import rikkei.academy.model.Customer;  
import rikkei.academy.model.Province;  
import rikkei.academy.service.customer.ICustomerService;  
import rikkei.academy.service.province.IProvinceService;  
  
import java.util.Optional;  
  
@Controller  
public class CustomerController {  
 @Autowired  
 private ICustomerService customerService;  
  
 @Autowired  
 private IProvinceService provinceService;  
  
 @ModelAttribute("provinces")  
 public Iterable<Province> provinces(){  
 return provinceService.findAll();  
 }  
  
  
 @GetMapping("/create-customer")  
 public ModelAndView showCreateForm() {  
 ModelAndView modelAndView = new ModelAndView("/customer/create");  
 modelAndView.addObject("customer", new Customer());  
 return modelAndView;  
 }  
  
 @PostMapping("/create-customer")  
 public ModelAndView saveCustomer(@ModelAttribute("customer") Customer customer) {  
 customerService.save(customer);  
 ModelAndView modelAndView = new ModelAndView("/customer/create");  
 modelAndView.addObject("customer", new Customer());  
 modelAndView.addObject("message", "New customer created successfully");  
 return modelAndView;  
 }  
  
// @GetMapping("/customers")  
// public ModelAndView listCustomers() {  
// Iterable<Customer> customers = customerService.findAll();  
// ModelAndView modelAndView = new ModelAndView("/customer/list");  
// modelAndView.addObject("customers", customers);  
// return modelAndView;  
// }  
 @GetMapping("/customers")  
 public ModelAndView listCustomers(@RequestParam("search") Optional<String> search, Pageable pageable ){  
 Page<Customer> customers;  
 if(search.isPresent()){  
 customers = customerService.findAllByFirstNameContaining(search.get(), pageable);  
 } else {  
 customers = customerService.findAll(pageable);  
 }  
 ModelAndView modelAndView = new ModelAndView("/customer/list");  
 modelAndView.addObject("customers", customers);  
 return modelAndView;  
 }  
 @GetMapping("/edit-customer/{id}")  
 public ModelAndView showEditForm(@PathVariable Long id) {  
 Optional<Customer> customer = customerService.findById(id);  
 if (customer.isPresent()) {  
 ModelAndView modelAndView = new ModelAndView("/customer/edit");  
 modelAndView.addObject("customer", customer.get());  
 return modelAndView;  
 } else {  
 ModelAndView modelAndView = new ModelAndView("/error.404");  
 return modelAndView;  
 }  
 }  
  
 @PostMapping("/edit-customer")  
 public ModelAndView updateCustomer(@ModelAttribute("customer") Customer customer) {  
 customerService.save(customer);  
 ModelAndView modelAndView = new ModelAndView("/customer/edit");  
 modelAndView.addObject("customer", customer);  
 modelAndView.addObject("message", "Customer updated successfully");  
 return modelAndView;  
 }  
  
 @GetMapping("/delete-customer/{id}")  
 public ModelAndView showDeleteForm(@PathVariable Long id) {  
 Optional<Customer> customer = customerService.findById(id);  
 if (customer.isPresent()) {  
 ModelAndView modelAndView = new ModelAndView("/customer/delete");  
 modelAndView.addObject("customer", customer.get());  
 return modelAndView;  
  
 } else {  
 ModelAndView modelAndView = new ModelAndView("/error.404");  
 return modelAndView;  
 }  
 }  
  
 @PostMapping("/delete-customer")  
 public String deleteCustomer(@ModelAttribute("customer") Customer customer) {  
 customerService.remove(customer.getId());  
 return "redirect:customers";  
 }  
}

Để việc hỗ trợ mapping tự động này xảy ra, cần cập nhật lớp ApplicationConfig, sử dụng annotation @EnableSpringDataWebSupport.

@Configuration  
@EnableWebMvc  
@EnableTransactionManagement  
@ComponentScan("rikkei.academy.controller")  
@EnableJpaRepositories("rikkei.academy.repository")  
public class AppConfig implements WebMvcConfigurer, ApplicationContextAware {  
 ...  
}

* Cập nhật file /customer/list.html để hiển thị các nút phân trang.

<!DOCTYPE html>  
<html lang="en" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
</head>  
<body>  
<a href="/create-customer">Create new customer</a>  
<h1>Customers</h1>  
<form method="get" action="/customers">  
 Search by first name:  
 <input name="search" type="text"/>  
 <input type="submit" value="Search"/>  
</form>  
<br/>  
<table border="1">  
 <tr>  
 <th>First name</th>  
 <th>Last name</th>  
 <th>Province</th>  
 <th>Edit</th>  
 <th>Delete</th>  
 </tr>  
 <th:block th:each="customer : ${customers}">  
 <tr>  
 <td th:text="${customer.firstName}"></td>  
 <td th:text="${customer.lastName}"></td>  
 <td th:if="${customer.province}" th:text="${customer.province.name}"></td>  
 <td th:unless="${customer.province}"></td>  
 <td><a th:href="@{/edit-customer/\_\_${customer.id}\_\_ }">Edit</a></td>  
 <td><a th:href="@{/delete-customer/\_\_${customer.id}\_\_ }">Delete</a></td>  
 </tr>  
 </th:block>  
</table>  
<div>  
 <a th:href="@{'/customers'(page=${customers.number - 1})}" th:if="${customers.hasPrevious()}">Previous</a>  
 <span th:text="${customers.number + 1}"></span>/<span th:text="${customers.totalPages}"></span>  
 <a th:href="@{'/customers'(page=${customers.number + 1})}" th:if="${customers.hasNext()}">Next</a>  
</div>  
</body>  
</html>

* Tạo lại config database và tạo database mới trong MYSQL

@Bean  
public DataSource dataSource() {  
 DriverManagerDataSource dataSource = new DriverManagerDataSource();  
 dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");  
 dataSource.setUrl("jdbc:mysql://localhost:3306/bai7\_th2\_province");  
 dataSource.setUsername("root");  
 dataSource.setPassword("Minhtri29092014");  
 return dataSource;  
}

Trong đoạn mã trên, đường link đến trang tiếp theo (Next) sẽ hiển thị nếu có trang tiếp theo. Đường link đến trang trước (Previous) sẽ hiển thị nếu có trang trước.

Chạy ứng dụng, thêm nhiều hơn 20 customer để quan sát kết quả.

https://github.com/nguyendongminhtri/MD4-Bai7-TH2-Page-Customer.git

Hướng dẫn nộp bài:

Up code lên github.

Paste link github vào phần nộp bài.