React-Spring-Pagination

by

Ronald Cook

www.linkedin.com/in/ronald-cook-programmer

September 2022

Overview of React-Spring-Pagination

Introduction

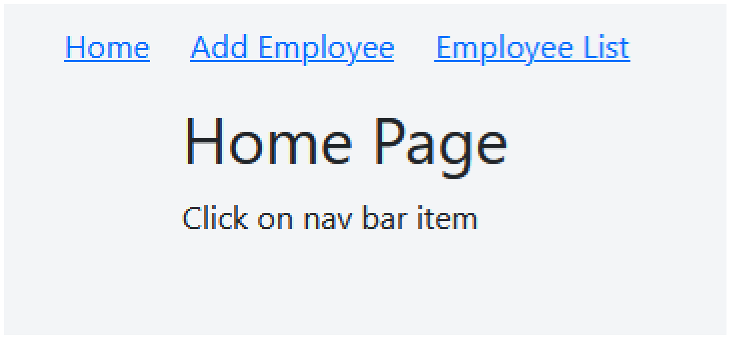
The purpose of this project is to learn some details about a simple application based on React, Spring Boot, REST API, and MariaDB database. React is used to create the UI front end, while Spring implements the back end. The application demonstrates pagination in React and Spring Boot. It is assumed you already know how to install the software. This is a desktop application which assumes a wide screen size, and both horizontal and vertical layouts are rendered.

Configuration

See the pom.xml and package.json files for the versions of each component in the application. Netbeans or another IDE may be used to build the jar file. The MariaDB database must be launched before this application. See the application.properties file under the resources folder for sample database configuration parameters. The sql file,sample-employdb.sql, may be used to populate the database with some initial records. The application.properties file also contains Hibernate and Spring configuration parameters. For Windows, a batch file, myReactBoot.bat, is provided to launch the application on port 8081, which may be set in resources/application.properties. Spring boot includes the tomcat server, and all application urls go to tomcat. React also includes a test server, but it is not used by this application. Open a browser and enter the URL, localhost:8081/front/home to open the home page.

Front end Details

The application consists of just 3 screens: Home, Add Employee, and Employee List.



Home Page

Click on nav bar item

The Home screen contains a horizontal navigation bar which is defined in Navbar.js, using an HTML list. To make the list horizontal, the CSS “nav > ul > li” in App.css includes “display: inline;”.

Also, note the “Link to” references in Navbar.js, which are defined within the Routes tag in App.js. Each Route tag associates a link path to a React component. In this case the components are Home, AddEmployee, EmployeeList, and NotFound, with associated links:

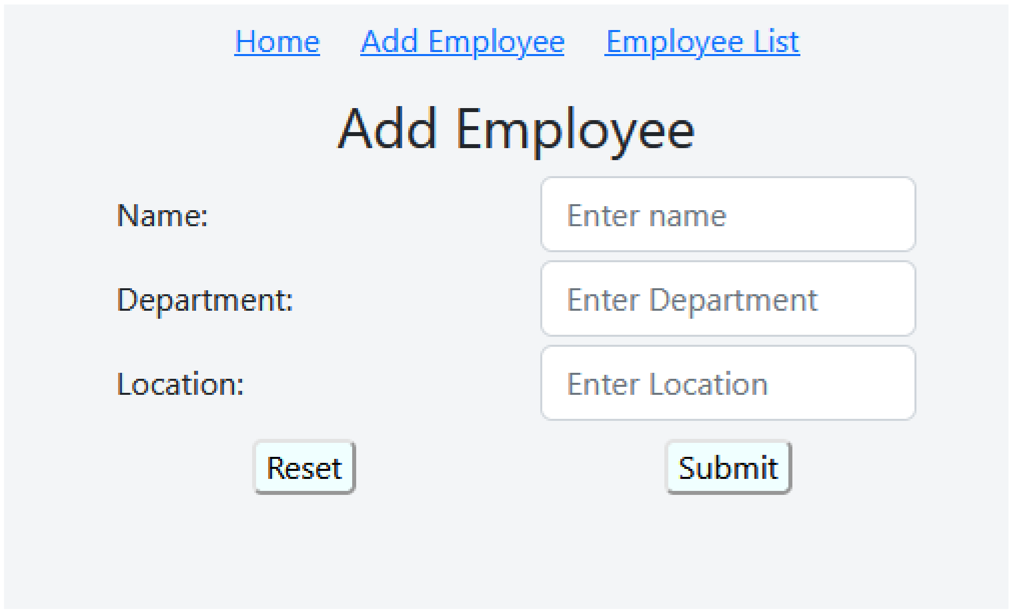
localhost:8081/front/home

localhost:8081/front/ add

localhost:8081/front/ employeeList

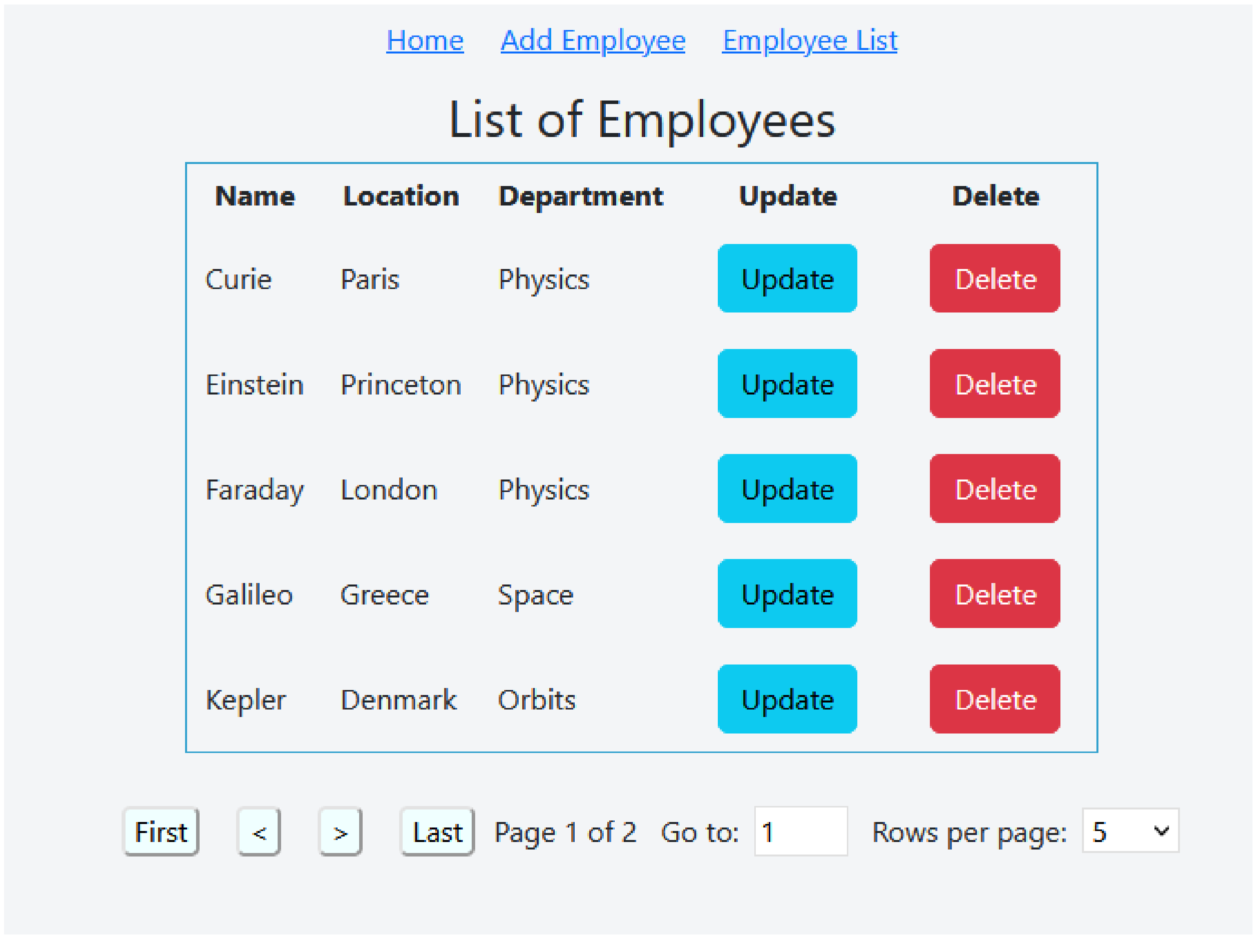
localhost:8081/front/ *

The Add Employee form contains input fields for Name, Department, and Location. Examine AddEmployee.js to see how the form is designed. Note how the form is centered horizontally with CSS “center App w400”. Text labels are aligned left with CSS “txleft”. Note the bootstrap CSS “col-lg-6” on the label tag and input field to create equal sized layout space.



The buttons are center aligned with “d-flex justify-content-center”. Reset is used to clear all form fields. Submit will send the form fields to the database via employeeService, which uses axios to convert the employee object into json for the Spring Boot REST service, explained later. If the user omits a required input, the submit halts, and an error message appears below the missing field. Look at the arguments to the register function on each form input field. If submit is successful, navigate returns to the Home screen.

The Employee List screen displays a table of employees, with additional columns to update or delete the data. Note this screen is created by two components: EmployeeList which prepares the column configuration, and PageTable which produces the actual HTML table. In EmployeeList note the function “useMemo” to prevent unnecessary calls on renders of the screen. Also, note the “cellProps” argument passed to the update/delete buttons. This represents the employee data in each row.



List of Employees

Name	Location	Department	Update	Delete
Curie	Paris	Physics	Update	Delete
Einstein	Princeton	Physics	Update	Delete
Faraday	London	Physics	Update	Delete
Galileo	Greece	Space	Update	Delete
Kepler	Denmark	Orbits	Update	Delete

Page 1 of 2 Go to: 1 Rows per page: 5

When the user presses the Update button, handleEdit is called with the employee argument, including the ID. The employee object is converted to json, and saved in sessionStorage for use in AddEmployee. The ID distinguishes a new employee from an updated employee. Therefore, the same form can be used to process both cases. AddEmployee checks sessionStorage for the update, and calls setValue to populate the form fields. Submit will call saveEmployee which invokes employeeService.update to send the updated employee to the backend.

The pagination buttons below the table allow the user to display the next page of data, previous page, or selected page. Paging requires a REST call to the backend.

Back end Details

The REST service is implemented by Spring Boot with PagingAndSortingRepository, to perform CRUD operations on a database. DBConfiguration sets up the database configuration from parameters in the application.properties file. EmployeeController maps REST paths to database calls. Axios and Spring Boot hide the conversion of EmployeeEntity to or from json, which is the format of the REST API.

Add employee sequence:

AddEmployee.save -> employeeService.create -> httpClient.post -> axios -> EmployeeController.saveEmployeeDetails -> EmployeeRepository.save -> database.

The url is: localhost:8081/back/employees, the http method is POST. The save to database performs a create operation because the data does not include an ID.

In the httpClient functions, note the url paths are enclosed in backticks, not quotes, to allow substitution variables.

Update employee sequence:

AddEmployee.save -> employeeService.Update -> httpClient.put -> axios -> EmployeeController.updateEmployee -> EmployeeRepository.save -> database.

The url is: localhost:8081/back/employees, the http method is PUT. The save to database performs a record update because the data includes an ID.

Employee List sequence:

EmployeeList.init -> fetchPage -> employeeService.getAll -> httpClient.get -> axios -> EmployeeController.getAllEmployees -> EmployeeRepository.findAll -> database -> PageData<EmployeeEntity> -> EmployeeList.fetchPage response -> setData, setPageCount -> PageTable -> cell.getCellProps.

The url is: localhost:8081/back/employees/1/5, and the http method is GET.

In EmployeeController.getAllEmployees, note the use of PathVariable rather than RequestParam, due to the structure of the path which puts the variables between slashes rather than after a question mark. In this example page = 1, and size = 5.

Employee List, next page(>) sequence:

PageTable.nextPage -> fetchData -> fetchPage -> (see above sequence).

The url is: localhost:8081/back/employees/2/5, and the http method is GET.

In the return from getAllEmployees, the PageData class includes page size and total pages as well as a list of EmployeeEntity. This is because total pages depends on page size, which the user may have modified before pressing the next or previous button. Total pages = (total rows in database) / (page size). Fortunately, findAll does this calculation for us, and returns the result in Page<EmployeeEntity>.

Conclusion

We have discussed a horizontal navigation bar, React form, React pagination table, axios http service, Spring

Boot REST API, and PagingAndSortingRepository. This small application has demonstrated many implementation details. Hopefully, you can reuse some of these features in building your own applications.