**HTML5-REST-Jakarta-Overview**

**by**

**Ronald Cook**

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

**February 2023**

**Overview of HTML5-REST-Jakarta**

**Introduction**

The purpose of this project is to learn some details about a simple application based on HTML5, javascript, json, REST, Jakarta, and Mysql. HTML5 and javascript are used to create the UI front end, while Jakarta implements the back end. The application demonstrates javascript web components, json processing, and SQL queries.

**Configuration**

See the pom.xml for the versions of each component in the application. Eclipse or another IDE may be used to build the war file. The Mysql database must be launched before this application runs. The sql file, simplemodels-dump.sql, may be used to populate the database with some initial records. All application urls go to the apache tomee plus server. Once the server is started, open a browser and enter the URL, localhost:8080/html5-jakarta-1.0/index.html to open the home page.

**Front end Details**

The application consists of just multiple screens: Home, Add Employee, Employee List, Add Customer, Customer List, and a series of Joins.

Graphical user interface, text

Description automatically generated

At the top of every screen there is a horizontal navigation bar which is defined in the javascript web component header.js, using HTML header and nav tags, and an HTML list. The CSS file, navDrop.css, provides spacing and color. At the bottom of every screen another navigation bar is defined in web component footer.js, which uses the HTML footer tag.

The Create Employee form contains input fields for Id, Name, Phone, Email, Report to, Job title, Address info. Examine employee-form.html 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.

Graphical user interface, text, application

Description automatically generated

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.

Graphical user interface, application

Description automatically generated

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.

Joins

Under the Joins tab, there are several join options to choose from. Each one demonstrates a basic SQL join operation.

**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 sample web application based on HTML5, REST, and Jakarta. On the front end, javascript web components have been created for header, footer, pager, and confirmer. The javascript fetch function has been used to send and receive json to and from the back end. Jakarta REST functions process the json seemlessly and interface with the mysql database using basic JDBC functionality. This small application has demonstrated many implementation details. Hopefully, you can reuse some of these features in building your own pplications.