**HTML5-REST-Jakarta Overview**

**by**

**Ronald Cook**

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

**March 2023**

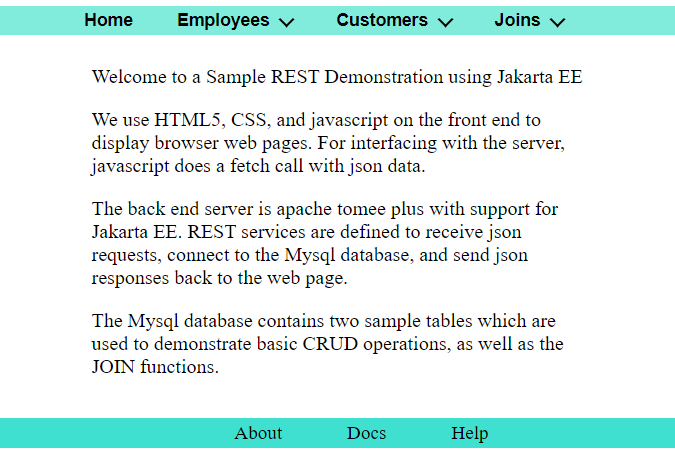
**Overview of HTML5-REST-Jakarta**

**Introduction**

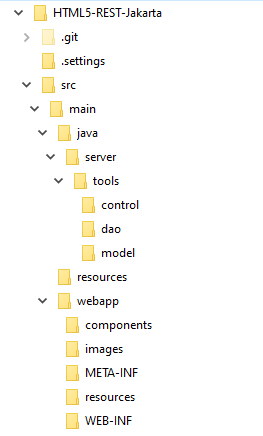
The purpose of this project is to learn some details about a simple application based on HTML5, javascript, json, REST, Jakarta, and MySql. The UI front end is created with HTML5, CSS and javascript, while Jakarta implements the back end. The application demonstrates javascript web components, json processing, 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 a server that implements the JakartaEE 10 API, such as Wildfly 27. Once the server is started, open a browser and enter the URL: localhost:8080/html5-jakarta-1.0 to open the home page, shown below.



The application code structure is shown here:



In the discussion that follows, it is most instructive to review the code to fully understand how it works.

**Front End Discussion**

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

**Header and Footer**

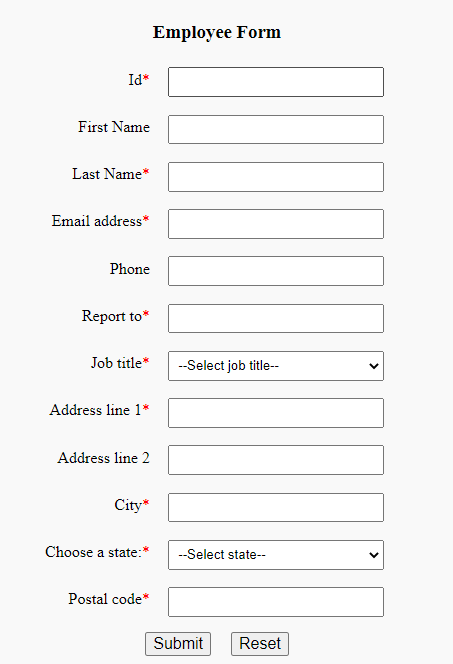
At the top of every screen there is a horizontal navigation bar which is defined in the javascript web component header.js, using HTML header and nav tags, and an HTML list. The CSS file, navDrop.css, provides spacing, color. At the bottom of every screen, another navigation bar is defined in web component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

**Employees**

Employees menu has two options.

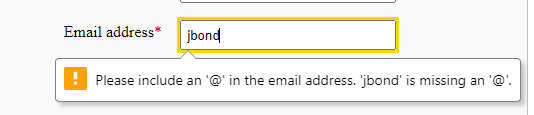


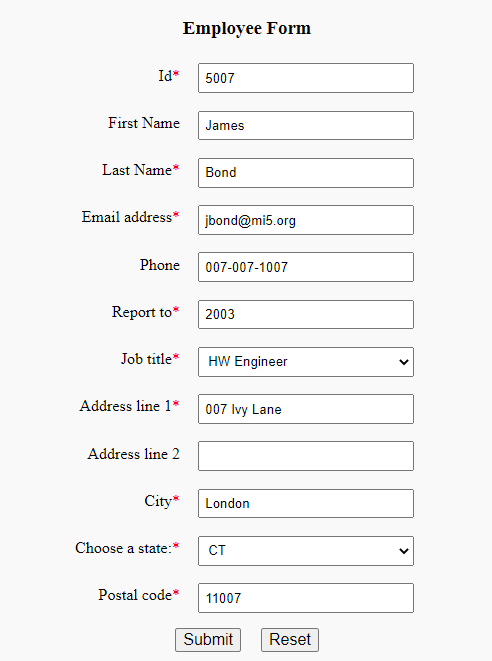
The Create Employee form contains input fields for Id, Name, Phone, Email, Report to, Job title, and Address info. Examine employeeForm.html and entryForm.css to see how the form is designed. Note how the form is centered horizontally with CSS “margin-left:auto” and “margin-right:auto”. The labels and input fields are grouped under a fieldset tag and arranged using “display: grid”.



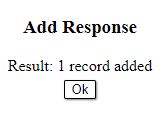
The buttons are center aligned with “display: flex” and “justify-content: center” CSS. The Reset button is used to clear all form fields. The Submit button will send the form fields to a POST request using a javascript fetch, in which the employee object is converted into json for the Jakarta REST service, explained later.

If the user omits a required input, the submit halts, and an HTML5 built-in form validation message appears below the missing input field.

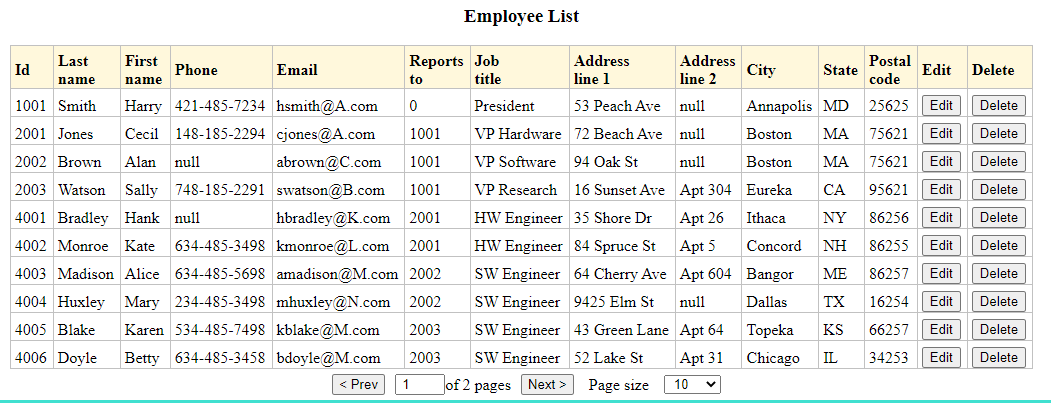




If submit is successful, a “notifier” web component appears showing the response from the server.

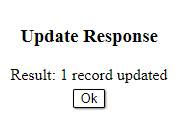


The Employee List screen displays a table of employees, with additional columns to edit or delete the data. At the bottom of this screen there is a row of page info, created by the pager.js web component. Note employeeList.html uses a table template, which is populated by javascript from listUtils.js. Paging data is sent to the REST server as a POST request to specify page size and page number to retrieve from the database. The response is a one page list of data to display. Each row is based on the Employee object, which is also passed to the edit/delete buttons.



When the user presses the Edit button, javascript is called with the employee object argument, which is converted to json, and saved in sessionStorage for use in employeeForm. The ID distinguishes a new employee from an updated employee. Therefore, the same form can be used to process both cases. However, in case of adding a new employee, a POST request is sent, whereas for the case of an update, a PUT request is sent.

If an update is successful, a “notifier” web component appears showing the response from the server.



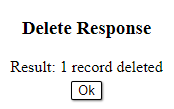
For a delete case, the “confirmer” web component pops up to allow the user to cancel:



When the user selects yes, the delete Id is sent to the REST service in a url:

http://localhost:8080/html5-jakarta-1.0/api/employees/5007

The “notifier” web component shows the result of the delete request.



The “pager” web component 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 back end via the url:

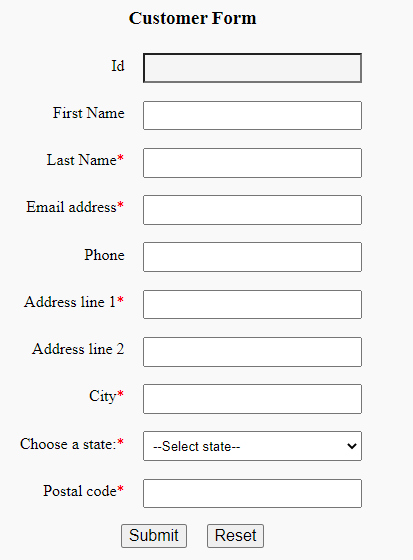
http://localhost:8080/html5-jakarta-1.0/api/employees/page

**Customers**

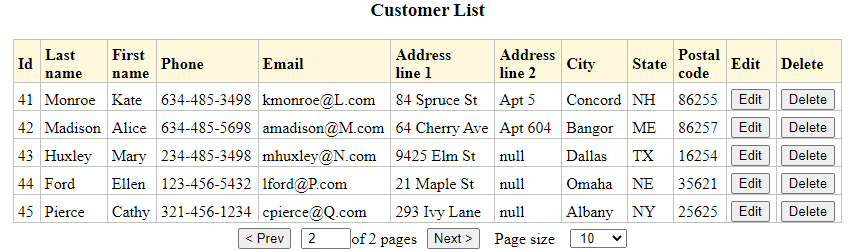
Customers menu has two options.



The Create Customer form contains input fields similar to the Employee Form above, and common functions are invoked from formUtils.js.



The Customer List displays a table similar to the Employee List described above, with shared functions located in listUtils.js.

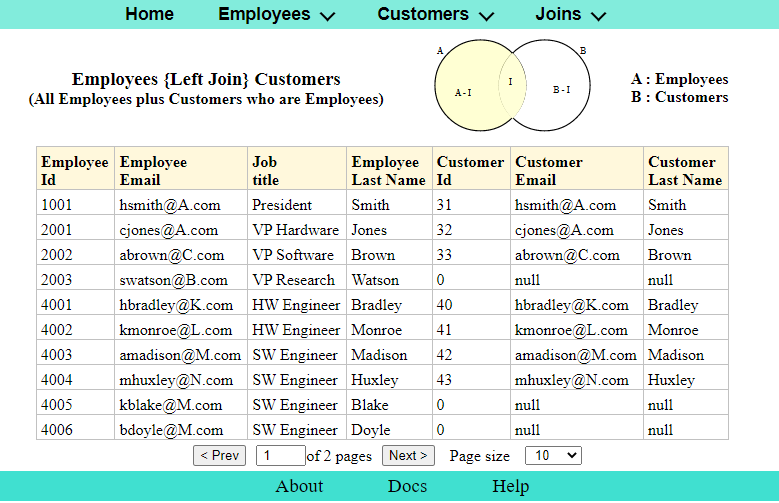


**Joins**

Joins menu has several options.



Each option demonstrates a basic SQL join operation between the employees database table and the customers table. The results are displayed in a Joins List table similar to Employees and Customers. For example, the left join list looks like the following:



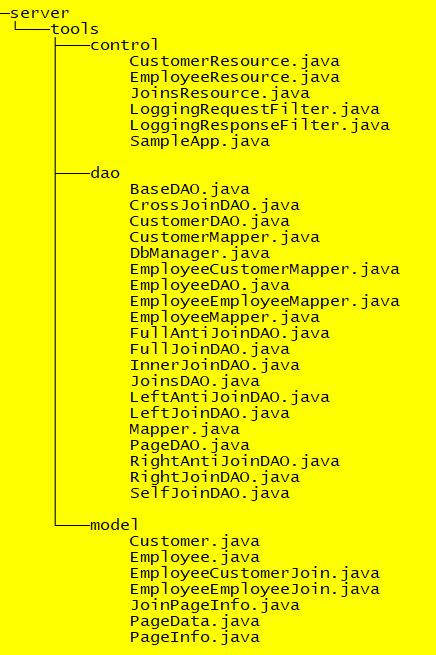
Because there are several join options, header.js calls the javascript function join when the user clicks on the option. The join function extracts the join type and join title from the html, and stores them in localStorage for use by joinList.html.

These items are then used to format the title, sub title, and image above the result list, as shown above. The join type must be included along with the page info in the REST request to the back end.

**Back End Discussion**

The REST service is implemented by Jakarta with JDBC, to perform CRUD and Join operations on a MySql database. DbManager.java sets up the database configuration parameters. EmployeeResource, CustomerResource, and JoinsResource map REST url paths to database access.

The back end code structure follows:



**Jakarta Application**

SampleApp extends the Jakarta Application class, and specifies the top level url path (api) for all REST services. EmployeeResource, CustomerResource, and JoinsResource each define additional url path suffixes to the top level. For example, the complete url path to display the Employee List is:

“http://localhost:8080/html5-jakarta-1.0/api/employees/page”

where html5-jakarta-1.0 is the name given to the war file.

**Employee Rest Service**

As an example, a REST request to get a list of employees goes through the LoggingRequestFilter, then to EmployeeResource, where getEmployeesByPage is called. Jakarta automatically parses the request and converts the json body into a PageInfo object, which is passed to EmployeeDao. There, two queries are constructed, one for selecting the list items, and another to retrieve the total count of the items in the database table.

This information and PageInfo are passed to PageDao<T>, where prepared statements are created and executed. The returned object has type PageData<T>, where T is the Employee class in this example. PageData contains an updated PageInfo, and list content specified by List<T>. But how is List<T> populated with the query results? List<Employee> is created with EmployeeMapper, which implements the interface Mapper<T>, by calling mapRow to convert the SQL resultSet into the Employee object.

By using a generic type T, PageDao, PageData, and Mapper are reusable classes. Back at EmployeeResource, PageData is added to the Response entity which is returned to the front end web page.

Note that in all the model classes, there are no setters and getters for simplicity. Since the conversions to and from json does not require setters and getters, there is no point in implementing them.

Also, note that Jakarta does support persistence frameworks, but here we simply used JDBC. For Jakarta Persistence API, see for example: https://itnext.io/whats-new-in-jakarta-persistence-3-1-by-examples-81b292e8b3a4, or http://www.mastertheboss.com/java-ee/jakarta-ee/jakarta-persistence-3-1-new-features/.

**Customer REST Service**

The CustomerResource and related classes are similar to EmployeeResource. They share the PageInfo, PageDao, PageData classes, while implementing custom Mapper classes.

**Joins REST Service**

The JoinsResource classes are a bit more complex due to the join type. For instance, when the left join page request is processed by JoinsResource, JoinsDao is called, which filters the join type in a case statement. From there, another dao is called, such as LeftJoinDao, which defines the customized queries for a SQL left join operation. Then the generic classes, PageDao, PageData, and Mapper, access the database as described above. Finally, in JoinsResource, PageData is added to the Response entity which is returned to the front end web page.

**Conclusion**

We have discussed a sample web application based on HTML5, REST, and Jakarta. On the front end, reusable javascript web components have been created for header, footer, pager, confirmer, and notifier. 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 your own applications.