Functionality documentation

Content

Last updated: 9th of December 2024 10:08, by Marius

Controller

The controller is what the program uses to communicate between the backend services and the frontend services, this is also where the function of all the different mappings are defined. I will explain shortly what the different mappings do and what type of data they take in.

list():
In: Nothing.
Function: Creates a list of all the employees in the database and shows it to the user.
Out: HTML displaying all the employees. (templates/employees/list-employees)

showFormForAdd():
In: Nothing.
Function: Creates an empty employee object which the user fills out with a form.
Out: HTML form for creating a new user. (templates/employees/employee-form)

showFormForUpdate():
In: An employee id.
Function: Connects an already existing employee to a form, allowing its data to be altered.
Out: HTML form for editing user. (templates/employees/employee-form)

showFormForDelete():
In: An employee id.
Function: Displays an employee in a deletion confirmation form.

Out: HTML confirming if the user wants to delete a specific employee (templates/employees/delete-form)

deleteEmployee():
In: An employee object.
Function: Deletes an employee from the database.
Out: Redirect to main page. ({domain}/employees/list}

saveEmployee():
In: An employee object.
Function: Saves the provided employee object to the database.

DAO (Data Access Object)

The DAO is what our application uses to communicate between the application itself (the logical part) and the database (the data part). This application uses something called JPA repository which does this automatically and eliminates the need to create code for database-application communication and eliminates the risk of us making errors in the code for said communication.

JPA repository **should** work with any database that you choose to use, but it is important to remember that just because it should, doesn't mean it does. If this is the case, then you're going to have to troubleshoot that issue yourself.

JPA repository also allows for custom methods, these are created simply by their name. For example: findAllByOrderByLastNameAsc(). This is a method that doesn't normally exist, but the program creates it when we define it in the interface. It does this by analyzing the name of the method:

findAll: Finds all the employees in the database.
ByOrder: Lets the program know that it's going to sort the list.
ByLastName: Tells the program to sort by last name.
Asc: Tells the program to sort ascending.

Entity

The entity is the object(s) that we're using as "rows" in the application. These objects are linked to the tables in the database and whenever we extract a row from the database it is converted to these objects.

Employee:
The employee class/object has four different variables which correspond to the four columns in the database (id, first_name, last_name, email).

Users:
This object has yet to be added, but is planned.

Service

The service layer is the layer that connects the JPA repository and activates many of its functions by implementing them. This is necessary because the JPA repository simply is an interface. These methods are basic and do exactly what their name say they do:

findAll() finds and returns all the employees in the database .

findById(int id) finds and returns a specific employee based on their unique id and throws an error if it doesn't find an employee with that id.

save(Employee employee) saves an employee object to the database, if the employee object has an id, then it will override the row In the database with that id and if it doesn't have one then the database will create a new row and give it a new id.

deleteById(int id) deletes the row from the database with the provided id.

Database

This database documentation will not go in depth into what exactly a database is and what it does as it is assumed that this is already known.

Employee table: The employee table holds the information about the employees in the repository and consists of 4 different columns:

id: id is the identifying column and contains a serial integer that is unique to each row of the database. This value can not be reassigned once assigned (except for manual editing) and is automatically generated and assigned when a new row is added to the table. This is a required column (not Null). This column is of the type **big serial**.

first_name: first_name is the column designated to storing the first (and potentially the middle) name of the employee, this column is **not** an identifying column and can therefore be the same for multiple rows. This column is not required (can be Null). This column is of the type **character varying**.

last_name: last_name is the column designated for the last name of the employee. This column is **not** an identifying column but is still required (not Null). This column is of the type **character varying**.

email: email is the column designated to storing the email of the employee. This column is **not** an identifying column but is still required (not Null). This column is of the type **character varying**.

Users table: The users table contains usernames and encrypted passwords that are used to log onto the website to perform different actions with different levels of authorization.

!!This table has yet to be added but is a planned feature!!

HTML

The HTML files are found under resources/templates

The HTML files are what the user sees in the browser. This application uses thymeleaf which allows for the transfer of data between the controller and the HTML file and for very basic logic in the HTML file itself. Beneath you'll find a description of what the different html files contain:

index.html:
The index.html file is what the program displays if you go to {domain}, here the file simply redirects to {domain}/employees/list.

employees/delte-form.html:
This html file displays the information about the employee with the id it is provided with hen it is shown by the controller and asks if the user really wants to delete this employee. If the answer is yes then it redirects to {domain}/employees/delete which deletes the user, and if the user chooses not to delete the user, then it redirects to {domain}/employees/list.

Employees/employee-form.html:
This is a form with multiple purposes, it can be used to both create a brand-new employee or to update an existing one. If the user is sent to this form by the showFormForAdd() method, then the form will be empty, and the user is free to fill in the information about a new employee they would like to add. If the user comes from showFormForUpdate() then the form will be pre-filled with the information about the employee, they wished to edit, and any changes made to the form will then be applied to the already existing employee when the user clicks save. The save button redirects to {domain}/employees/save.

employees/list-employees.html:
This form takes the employee list it is provided with by the controller and displays it as a table for the user to see, this form also contains buttons rerouting the user to the other functions of the program.