Code Documentation for FLW App

(Information for code & database as of 20 Oct 2017)

Contents

Application Module Files

This applies to the following files, all found in the folder “js”:

datainput.js

hotspot.js

initlialisation.js

normalisation.js

These files are for creating the AngularJS modules used in the application, as well as to include the dependencies required by them.

For example, in the file “initialisation.js”, it declares a module named “initialisation”, and that it requires a module named “angular-js-xlsx”.

Controllers

These files are located in the “js/controllers” folder.

controller.datainput.js

This file is the controller for the datainput module, and is used for adding entries via spreadsheet upload and directly through the dropdown list. Upon load, this controller first watches the variable “$scope.data.selectedProcess” (note that “$scope” is omitted from the actual string). This variable is the user’s choice in the dropdown list on the corresponding HTML page (entry.html). When the selected process is changed, so may its number of inputs, outputs and wastes. Hence, the arrays containing amounts for the inputs, outputs and wastes are reset when the selection changes, and these arrays are populated with the correct number of entries based on the amount of inputs, outputs and wastes of the selected process.

Next, a request for the process information from the database is sent, and the environment is initialised.

initEnviron()

Parameters: -

Returns: -

This function initialises the variables used by this controller and the corresponding HTML page.

getProcesses()

Parameters: -

Returns: -

This function sends a GET request to the PHP server to fetch process information from the database. When the response is received, the dropdown list is populated based on the results, and the environment is initialised.

populateProcessesList(processes)

Parameters:

- processes, an array containing the information on the processes that were initialised in the system

Returns: false if the PHP response was invalid

This function goes through the fetched processes and prepares them for selection by the user through the dropdown list.

For every new process encountered, an object is created, containing its name, and 3 arrays for its inputs, outputs and wastes. This object is pushed into the array processesList, while the process name is added into the array addedProcessNames (this array is for checking whether a process has already been added, and for getting its index). Each of these process objects are then populated with their associated inputs, outputs and waste materials.

These process objects are then sorted by name in alphabetical order. This sorting is also done for each of the arrays within each process object.

compareProcessesByName (a, b)

Parameters: a and b, the 2 items to be compared when sorting.

Returns: -

This function is a custom sort function for sorting objects according to their name property, in alphabetical order.

compareArrayEntriesByName(a, b)

Parameters: a and b, the 2 items to be compared when sorting.

Returns: -

This function is a custom sort function for sorting strings in alphabetical order. It is used in the populateProcessesList function.

$scope.submitData()

Parameters: -
Returns: false if invalid values are detected.

This function is called when the “Submit Entry” button in entry.html is clicked. It first checks whether the date entered by the user is valid. If not, the function returns prematurely with a value of false.

Similarly, the amount values entered by the user for inputs, outputs and waste are checked for validity as well.

If everything is valid, a POST request is sent to the PHP server, along with the entries to be added. After the response is received, the result is alerted to the user and logged in the console.

In the process of uploading, the status is also monitored by $scope.appEntriesStatus. This affects whether to display text informing user of the uploading on the HTML page.

parseDate()

Parameters: -
Returns: split, an array containing day, month, and year, in that order.

This function tries to split the user-entered data string based on the “/” character, and checks whether it is valid. If so, an array containing the day, month and year is returned. Otherwise, false is returned.

prepareEntriesFromInput(date, amountArray, IOW)

Parameters:

- date, an array containing day, month and year values
- amountArray, an array containing the amounts of materials to add
- IOW, a string indicating whether the amountArray contains input, output, or waste materials.

Returns: boolean representing whether all entries were valid and prepared successfully

This function is to prepare objects containing entry information to be added into the database. It loops through the amountArray, and if any value is invalid, the function returns with false. Otherwise, the respective material name is fetched from the selected process’ information.

Then, an object containing the dates, role of material, process name, material name, amount, and remarks is created. This object is finally pushed into the array $scope.entriesToAdd. (note: remarks are currently used for the data collector’s name. This function is used for uploading of data directly through the dropdown list, and a value of “app” is used instead for remarks here)

If everything is successfully, the function returns with a value of true.

$scope.readEntries(workbook)

Parameters:

- workbook, an excel spreadsheet that was uploaded (.xls or .xlsx)

Returns: -

This function is called when the browse button in entry.html is clicked, to parse the uploaded spreadsheet. If checks the spreadsheet for validity while adding them to an array. If everything is valid, it then calls a function to add the entries into the database.

$scope.error(e)

Parameters:

- e, an error thrown and caught by the spreadsheet parser library used.

Returns: -

This function is for handling the error encountered when attempting to upload a spreadsheet. Currently, the error is logged in the console and alerted to the user.

addSpreadsheetEntries(workbook)

Parameters:

- workbook, an excel spreadsheet that was uploaded (.xls or .xlsx)

Returns: boolean representing if the addition was successful

This function checks whether the uploaded spreadsheet and its entries are valid, and adds them into an array for uploading into database.

Note: The code parses the spreadsheets until the detected last row index. It expects every row to be valid / complete, otherwise the current implementation for uploading information via spreadsheet will reject it and alert the user. However, there are times when a false last row value is used. This can happen when we empty the cells of the last row(s) of the spreadsheet, without actually deleting the rows. It seems that these rows are still stored as present in the spreadsheet, with empty cells. As a result, the code will attempt to parse these extra row(s) and reject the spreadsheet. To fix this, highlight the residue row(s), right click, select "Delete" and "Entire row".

The code loops through each row of the spreadsheet, and creates an object representing the entry to be added. These objects are then added into the array $scope.entriesToAdd.

parseDateString(string)

Parameters:

- string, a string retrieved from the uploaded spreadsheet's date cell

Returns: date, a JavaScript date object

This function uses the date string from the spreadsheet to create a JavaScript data object and returns it.

addEntriesToDatabase()
Parameters: -
Returns: -

This function sends a POST request to the PHP server along with the prepared data entries, for adding them into the database. The decision to replace the last uploaded entries will determine which PHP file is involved.

After the response is received, the status of the request is alerted to the user.

controller.hotspot.js

This file is the controller for the hotspot module. Upon load, it fetches the available timeframe choices based on the database entries and initialises the environment.

hotspotCallback()
Parameters: -
Returns: -

This s a callback function to be passed into the function that begins hotspot analysis. This is required due to asynchronous execution. The function gets the required information from the hotspot service, and uses them to draw the hotspot analysis chart.

$scope.beginHotspots()
Parameters: -
Returns: -

This function is called when the button to begin hotspot analysis in wasteHotspots.html is clicked. It calls hotspotService to commence hotspot analysis.

$scope.showTotalWeight()
Parameters: -
Returns: float of the total waste amount, rounded if required.

This function is used in wasteHotspots.html to format how the total waste amount is shown. If the amount is not a whole number, it is rounded to decimal places. The amount is then returned.

prepareChartData(data)
Parameters: data, an array containing the processes involved in the hotspot analysis.
Returns: data, the passed parameter, after it has been processed.

Note: this function is used for drawing the Pareto Chart using the d3 library. The current version uses kendo, so this function can be removed.

The function sorts the processes based on their waste amounts in descending order. It then adds the properties cumulativeAmount and cumulativePercentage to each of them. If there are more than 30 processes, then 31st and subsequent processes are combine into 1 process, named “Others”.

getTimeframe()
Parameters: -
Returns: -

This function is for retrieving the timeframe choices available to the user. It sends a GET request to the PHP server. After the response is received, the possible choices are added into the arrays $scope.monthsData and $scope.yearsData. Finally, the environment is initialised.

initEnviron()
Parameters: -
Returns: -

This function initialises the variables required by this controller.

controller.initialisation.js

This file is the controller for the initialisation module. Upon load, it calls a function to initialise the environment.

initEnviron()
Parameters: -
Returns: -

This function initialises the variables used in this controller,

$scope.readMatrix(workbook)

Parameters:

- workbook, an uploaded excel spreadshseet (.xls or .xlsx)

Returns: -

This function is called when the browse button in initialisation.html is used to upload a spreadsheet containing skeleton and process information. It calls a function to add the entries into arrays and if successfu, it calls another function to add the entries into the database.

$scope.error(e)

Parameters:

- e, an error thrown and caught by the spreadsheet parser library used.

Returns: -

This function is for handling the error encountered when attempting to upload a spreadsheet. Currently, the error is logged in the console and alerted to the user.

addSpreadsheetMatrixAndProcessInfo(workbook)

Parameters:

- workbook, an uploaded excel spreadshseet (.xls or .xlsx)

Returns: a boolean representing whether the spreadsheet cells were added successfully.

This function is for parsing the spreadsheet containing matrix skeleton and process information, and add them into arrays for adding into the database. Refer to the formats of the spreadsheets for easier comprehension.

For process information, an object is created for each process. Each of these objects will subsequently be added with objects representing their input, output and waste materials.

For matrix skeleton, an objet is created for each matrix cell. Each of these objects will contain the row index, column index and the cell value.

If at any point, an entry is invalid, the entire spreadsheet is rejected and the function returns false.

addMatrixAndProcessesToDatabase()

Parameters: -
Returns: -

This function is for adding the parses spreadsheet entries into the database. It sends a POST request to the PHP server, along with the arrays containing the process information and matrix skeleton cells.

When the response is received, the user is alerted with a message according to the operation's success.

checkValidInput()

Parameters: -
Returns: a boolean representing whether the amounts entered by the user for saving average costs are valid.

This function serves to only allow saving of average costs if the user entered positive numbers (0 is allowed. Negative numbers and non-numbers are disallowed).

$scope.addAverageCosts()

Parameters: -
Returns: false if the user-enter values are not valid.

This function is called when the button for saving average costs in initialisation.html is clicked. It first checks if the amounts are valid, and returns prematurely with false if invalid. Otherwise, a POST request is sent to the PHP server along with the user-entered cost amounts.

When the response is received, the user is alerted of the results.

controller.normalisation.js

This file is the controller for the normalisation module. Upon load, it fetches the available timeframe choices based on the database entries and initialises the environment.

getTimeframe()

Parameters: -
Returns: -

This function is for retrieving the timeframe choices available to the user. It sends a GET request to the PHP server. After the response is received, the possible choices are added into the arrays $scope.monthsData and $scope.yearsData. Finally, the environment is initialised.

initEnviron()
Parameters: -
Returns: -

This function initialises the variables used by this controller.

$scope.buildMatrixSkeleton()
Parameters: -
Returns: -

This function is called when the “Confirm Timeframe” button in wasteIntensity.html is clicked. It sends a GET request to the PHP server to fetch the entries pertaining to the matrix skeleton. If there are such entries, the function returns prematurely.

After the response is received, it will create the matrix in a 2D array, where the primary arrays are the row, and the secondary arrays are the columns ([[row][col]]). Each matrix cell is represented by an entry object, each having a value property and a type property.

After the matrix skeleton is constructed, a function to retrieve the data entries is called.

createEntry(cellValue, materialType)
Parameters:
- cellValue, a float containing the value of a matrix numeric cell.
- materialType, a string representing whether this cell is an input, output, waste material, or an empty cell.

Returns: entry, a numeric entryObject

This function calls the appropriate entryObject constructor based on whether this cell is an input, output, waste, or empty cell. It checks for cell values that are negative (input), or zero (empty). If positive, it checks whether the passed parameter materialType is “Waste”, otherwise this is an output cell.

entryObject(value, type)
Parameters:
- value, a float representing material amount
- type, a string representing representing whether this cell is an input, output, waste material, or an empty cell.

Returns: itself

This is a constructor function for objects representing each matrix cell. It stores the 2 passed parameters in its own value and type properties.

getEntries(selectedTimeFrame, selectedYear, selectedMonth)

Parameters:

- selectedTimeFrame, a string representing the data entry timeframe chosen by the user.
- selectedYear, an int representing the year selected by the user, if not the default value.
- selectedMonth, a string representing the month selected by the user, if not the default value. Contains name of month & year (year is required since e.g. May 2016 is not the same as May 2017).

Returns: -

This function fetches the data entries from the database according to the timeframe chosen by the user. It sends a GET request to the PHP server along with the timeframe information.

When the response is received, it calls a function to add the entries into the matrix skeleton. If this succeeds, it proceeds on to create a balanced matrix and presents the user with output material choices (normalisation target).

addEntriesToMatrix(entries)

Parameters:

- entries, an array of the data entries fetched from the database.

Returns: a boolean representing whether the addition of the entries into the matrix skeleton was successful.

This function adds the data entries fetched form the database into the matrix skeleton. If the data entry's material or process names are not found within the matrix skeleton, the procedure is rejected and the function returns with false, while alerting the user of the error.

Otherwise, the numeric value of the entry is added to the respective cell. Note that values representing inputs are positive in the database, and are negated before being added into the matrix.

If the target matrix cell was previously empty, its type property is changed to input, output or waste accordingly.

Finally, the matrix skeleton is trimmed to remove all empty rows and columns and stored in $scope.inputMatrix.

trimMatrix(matrix)

Parameters:

- matrix, a 2D array representing the source matrix to be trimmed.

Returns: trimmedMatrix, a matrix cloned from the passed parameter, but with empty rows & columns excluded.

This function creates a new 2D array based on the passed parameter. It first goes through every row and copies the entire row to a temporary matrix only if it is not entirely empty (an empty row = all values are 0).

Next, it goes through every column and looks for columns that are entirely empty, and the index of these columns are stored in the array columnsToDrop. The function then goes through every row in the temporary matrix and adds all entries whose column index were not marked for exclusion.

This trimming must be done otherwise the matrix will not end up balanced, and consequently has no inverse. The trimmed matrix is finally returned.

createBalancedMatrixAndAllocate()

Parameters: -

Returns: -

This function begins the balancing procedure for squaring the input matrix, so that an inverse can be determined. It extracts one column of the input matrix one at a time, and adds it into $scope.balancedMatrix. If the column (which represents a process) has more than 1 output material, a function is called to perform allocation on it first, before adding it into $scope.balancedMatrix.

allocateProcess(allocationSource, numOfOutputs, totalOutputAmount, outputIndexes)

Parameters:

- allocationSource, an array representing the process (column) that needs to be allocated.
- numOfOutputs, an int of the number of output materials in this process.
- totalOutputAmount, a float of the combined amount for all outputs of this process.
- outputIndexes, an array containing the indexes of the outputs within this process (column).

Returns: -

This function goes through every row of the passed allocationSource column, and calls the respective allocation functions depending on the cell type.

allocateProcessName(processName, numOfOutputs)

Parameters:

- processName, a string of the process name
- numOfOutputs, an int of the number of output materials in this process.

Returns: -

This function splits the single process name cell into n cells, where n = numOfOutputs, as per the allocation methodology. The result process names will be appended with "(x)", where x begins from 1.

These cells are then pushed into the first row of $scope.balancedMatrix.

allocateInputEntry(numOfOutputs, totalOutputAmount, outputIndexes, allocationSource, row)

Parameters:

- numOfOutputs, an int of the number of output materials in this process.
- totalOutputAmount, a float of the combined amount for all outputs of this process.
- outputIndexes, an array containing the indexes of the outputs within this process (column).
- allocationSource, an array representing the process (column) that needs to be allocated.
- row, a int representing the row index of this cell that is undergoing allocation

Returns: -

This function splits the single input cell into n cells, where n = numOfOutputs, as per the allocation methodology. The input amount is split across the cells according to the mass ratio of the associated output's mass.

These cells are then pushed into the passed row of $scope.balancedMatrix.

allocateOutputEntry(numOfOutputs, outputStaggerCount, row, entrySource)

Parameters:

- numOfOutputs, an int of the number of output materials in this process.
- outputStaggerCount, an int representing the row which the the output amount should be placed.
- row, a int representing the row index of this cell that is undergoing allocation.
- entrySource, float representing the output amount that is being allocated.

Returns: -

This function splits the single output cell into n cells, where n = numOfOutputs, as per the allocation methodology. If the added column index matches the passed outputStagger count, the passed value of entrySource is used for this cell, otherwise an entryObject with the value of 0 is used.

The aim of this function (when run through all outputs of a column) is to convert $\begin{pmatrix} 2 \\ 2 \end{pmatrix}$ into $\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$.

These cells are then pushed into the passed row of $scope.balancedMatrix.

allocateWasteEntry(numOfOutputs, totalOutputAmount, outputIndexes, allocationSource, row)

Parameters:

- numOfOutputs, an int of the number of output materials in this process.
- totalOutputAmount, a float of the combined amount for all outputs of this process.
- outputIndexes, an array containing the indexes of the outputs within this process (column).
- allocationSource, an array representing the process (column) that needs to be allocated.
- row, a int representing the row index of this cell that is undergoing allocation

Returns: -

This function splits the single waste cell into n cells, where n = numOfOutputs, as per the allocation methodology. The waste amount is split across the cells according to the mass ratio of the associated output's mass.

These cells are then pushed into the passed row of $scope.balancedMatrix.

allocateEmptyEntry(numOfOutputs, row, entrySource)

Parameters:

- numOfOutputs, an int of the number of output materials in this process.
- row, a int representing the row index of this cell that is undergoing allocation
- entrySource, float representing the cell amount that is being allocated.

Returns: -

This function splits the single empty cell into n cells, where n = numOfOutputs, as per the allocation methodology. Each of the new cells will have the passed entrySource value (0).

These cells are then pushed into the passed row of $scope.balancedMatrix.

finishBalancing()

Parameters: -
Returns: -

This function continues the balancing procedure on $scope.balancedMatrix. The function goes through every row of the matrix, and checks whether the material in that row is a waste, input, output or intermediary material. If it is an intermediary material, the type cell of the material is replaced with a new entryObject that reflects this.

If the material is an input or waste, it calls a function to add an accounting column for the material into the matrix.

If the material is an output or intermediary material, their row index is then used for adding a normalisation candidate.

addAccountingColumn(materialIndex, amount)

Parameters:

- materialIndex, an int of the row index of the material that needs accounting for.
- amount, a float of the value to be used for the accounting

Returns: -

This function adds an accounting column to the end of $scope.balancedMatrix. This added column will have the name of the material being accounted for, appending with “<acc>”. The added column will be filled with zeroes, except for the row of the material being accounting for, denoted by the parameter materialIndex. At this row, the passed amount is used instead.

optionObject(type, name, index)

Parameters:

- type, a string representing whether this material is an output, intermediary etc.
- name, a string of the material's name.
- index, an int representing the row index of the material

Returns: itself

This is a constructor function for objects representing normalisation choices for the user to choose. Each will contain properties with values of the passed parameters.

addNormalisationCandidate(row)

Parameters:

- row, an int representing the row index of the material

Returns: -

Creates an optionObject for normalisation choice, and adds it into the array $scope.unitOptions.

customSortUnitOptions()

Parameters: -

Returns: -

This is a custom sorting function that sorts the normalisation material options. It will sort the options such that outputs appear on top of the list, while intermediary materials are ordered below. Within each group, the materials are ordered in alphabetical order.

This function currently sorts waste materials as well, which will be placed below intermediary materials. However, the current implementation of the app will not include waste materials among the normalisation choices in the first place.

$scope.keydownFunction(event)

Parameters:

- event, an event handler for triggering other procedures.

Returns: -

This is function is just for calling the startNormalisation function when the enter key is pressed in the normalisation target unit box.

extractSubmatrix(matrix)

Parameters:

- matrix, a 2D array of a matrix.

Returns: subMatrix, a 2D array that is a sub-matrix of the passed matrix.

This function creates and return a new matrix from the passed matrix, such that the new matrix contains only values of the numeric cells (the headings, process names, material names, type are excluded). This matrix will be used for inverse calculation.

$scope.isValidChoice(model)

Parameters:

- model, an object that binds the user's normalisation choices.

Returns: a boolean representing whether valid normalisation options are selected.

This function checks whether the user has selected a material to normalise, as well as whether the user have entered a positive number. This is for disabling the normalisation button if the conditions are not met.

$scope.startNormalisation()

Parameters: -

Returns: -

This function is called when the normalise button on wasteIntensity.html is clicked. It first calls the function to extract the sub-matrix from the current balanced matrix, and attempts to find its inverse. If an inverse does not exist (e.g. the matrix was balanced incorrectly), an alert is shown to the user and the function returns prematurely.

Otherwise, the function initiates the other procedures for the normalisation procedure.

createDemandVector()

Parameters: -

Returns: -

This function creates the demand vector based on the user's selected choices. The result is an array filled with 0s, except for the index corresponding to the user's selected material's row. This index is filled with the user-entered normalisation amount instead. This array is then stored in $scope.demandVector.

getScalingVector()

Parameters: -

Returns: -

This functions creates the scaling vector by multiplying the inverse with the demand vector. This function uses the numeric library. The result is is stored in $scope.scalingVector.

scaleSubmatrix()

Parameters: -
Returns: -

This function starts creating the normalisation results, by scaling each column of the sub-matrix with the corresponding scaling vector. Each of the entry in the sub-matrix are replaced with the normalised value.

buildNormalisedMatrix()

Parameters: -
Returns: -

This function builds the normalised matrix, by combining the text entries (process names, headings, etc.) from the balanced matrix, along with the normalised numeric entries of the normalised sub-matrix. The result is stored in $scope.normalisedMatrix.

trimNormalisedMatrix(matrix)

Parameters:

- matrix, a 2D array representation of a matrix

Returns: trimmedMatrix, a 2D array representation of a matrix

This function is used to clone the normalised matrix, while dropping all empty rows and columns, as well as the accounting columns. The result is then returned, which is used for displaying a compact and relevant matrix to the user, should the user wishes to view the matrix.

setResultsList()

Parameters: -
Returns: -

This function is for creating the appropriate strings for showing the user of the normalisation results. First it creates a string based on the user's choice of normalisation material and amount, which is then stored in $scope.data.resultsHeading.

The function then goes through each row of the trimmed normalised matrix, and looks for materials that serve as input or waste with respect to the normalisation target material. If such a material is found, an object is created that contains the material name, and an amount property. It then goes through that material row and gathers to total amount of that material involved (input values are negated because input amounts are negative in the matrix).

These objects are then pushed into $scope.data.resultsInput and $scope.data.resultsWaste accordingly. The current implementation does not count the input portions of intermediary materials. Only the "raw inputs" that have to be added into the system is recognised as input.

$scope.formatEntry(entry)

Parameters:

- entry, a string or float, representing the value of a matrix cell.

Returns: roundedEntry, a string or a float, depending on the type of cell value being formatted.

This function is for formatting the entries in the matrix shown to the user. If the passed entry is for a text cell (e.g. process name, material name, etc.), the value is returned as it is.

Otherwise, it is a float, and is rounded to 3 decimal places. If the rounded value is 0, an empty string is returned so that the matrix will not show a bunch of 0s, for a cleaner look. Otherwise, the rounded value is retuned.

$scope.startExport()

Parameters: -

Returns: -

This function is called when the export button in wasteIntensity.html is clicked. It begins a series of procedures to export the normalisation results into a spreadsheet. The spreadsheet will have the following structure:
<Original Input> <Demand Vector> <Scaling Vector>
<Balanced Matrix>
<Normalised Matrix>

changeEntryObjectsToValues(matrix)

Parameters:

- matrix, a 2D representation of matrix, contining entryObjects in its cells.

Returns: newMatrix, a 2D representation of matrix based on the passed matrix

This function creates and returns a new matrix based on the passed matrix. The passed matrix contains entryObjects in each of its cells, while the new matrix contains the actual value of each corresponding cells.

addFirstRowOfMatrix(exportMatrix)

Parameters:

- exportMatrix, a 2D representation of a matrix that will contain the information to be shown in the exported spreadsheet.

Returns: -

This function adds the cells of the input matrix, demand vector and scaling vector into the exportMatrix, as well as headings for them. Each of them are separated by an empty column. An empty row is added at the end for spacing with the matrix that will be added below.

addBalancedMatrix(exportMatrix)

Parameters:

- exportMatrix, a 2D representation of a matrix that will contain the information to be shown in the exported spreadsheet.

Returns: -

This function appends the cells of the balanced matrix into exportMatrix, as well as a heading for it. An empty row is added at the end for spacing with the matrix that will be added below.

addNormaliseMatrix(exportMatrix, normalisationAmount, normalisationMaterial)

Parameters:

- exportMatrix, a 2D representation of a matrix that will contain the information to be shown in the exported spreadsheet.
- normalisationAmount, the normalisation amount specified by the user.
- normalisationMaterial, the normalisation target material specified by the user.

Returns: -

This function appends the cells of the normalised matrix into exportMatrix, as well as a heading for it.

createWorkbookAndSave(exportMatrix, filename)

Parameters:

- exportMatrix, a 2D representation of a matrix that contains the information to be shown in the exported spreadsheet.
- filename, a string of the filename for the exported spreadsheet

Returns: -

This function adds the exportMatrix into a worksheet, which is then added into a workbook. The workbook is then offered as a download to the user.

Directives

This file is located in the “js” folder.

directives.js

This section applies only to the “paretoChart” directive in this file. It is for drawing the Pareto Chart using the d3 library.

Note: the “paretoChart” directive is no longer used because the chart is now no longer drawn by the d3 library. Feel free to delete this section (and the relevant code).

"paretoChart" directive:

Internal functions:

setChartParameters()
Parameters: -
Returns: -

This function initialises various chart parameters. First of all, the code removes all svg elements from the page, so that previously drawn charts do not remain on screen.

Next, the axes of the chart and the scales used are initialised. The x-axis uses the id of each process for its domain, while the left y-axis uses the waste amount of each process for its domain. The right y-axis, named yAxis2, uses the cumulative waste amount for its domain.

Following that, the dotted line of the Pareto Chart is initialised. The tooltip is then initialised as well. An offset is set for the tooltip of the first process/bar, in order to prevent it from crossing the left edge of the screen.

getBarColour(id)
Parameters:

- id, an int representing the id number of a specific process

Returns: a string representing the colour to be used with that process's bar

This function compares the id of a process with the number of hotspots identified. If the id is less than or equal to it, this process is a hotspot. The corresponding colour string is then returned. Currently, dark red is used for hotspots, while steel blue is used for the other processes.

External functions:

scope.drawChart()
Parameters: -
Returns: -

This function is for drawing the Pareto Chart. Note, this function is exposed to the parent scope due to the use of "scope." in the function name. This is currently used by the hotspot controller, where it calls this function using "$scope.drawChart()".

The function first initialises the required chart parameters, and starts adding each chart element into the svg object, along with their respective attributes. Some level of knowledge on the d3 library is recommended to fully understand this function.

Services

The files in this section are located in the “js” folder.

hotspotService.js

This file is for a service that will perform hotspot analysis, and is used by the hotspot controller.

Internal functions:

addEntries()

Parameters: -
Returns: -

This is a function to go through all data entries (that were already fetched into this service), and compile the relevant information per process, as required for hotspot analysis.

In the loop, entries corresponding to input materials are ignored since hotspot analysis is only interested in waste & output.

Two arrays are maintained: gatheredProcesses and gatheredProcessNames.
gatheredProcesses is an array that will ultimately hold process objects, and is used for hotspot analysis.
gatheredProcessNames is an array of strings, for recording & indexing the name of processes that have already been added into the gatheredProcess array.

Upon first encounter of a process name, an object is created, which contains process name, total waste amount, an array of waste materials, an array of individual waste amounts, and the total output amount. When a data entry with a process name that has already been added is encountered, its index is fetched from gatheredProcessNames, which is also the index of the corresponding object in gatheredProcesses. The object’s information is then updated.

A similar way is used to record & index waste materials encountered for each process. This is done because individual waste amounts per process is required by the measures recommendation component.

roundAndNormaliseWasteAmount(normalise)

Parameters:

- normalise, a boolean representing whether the hotspot analysis should be done with the waste amounts normalised (waste divided by output).

Returns: -

This function is to go through each of the process objects in the gatheredProcesses array, and round their waste amount to 2 decimal places if they are not whole numbers.

If the parameter "normalise" is true, then the waste amount is first normalised, by dividing it over the output amount.

assignIDtoProcesses()

Parameters: -
Returns: -

This function assigns a string ID to each process object in gatheredProcesses, to be used as the x-axis labels in the Pareto Chart.

The process objects are first sorted according to their waste amount, in descending order. Starting from the largest waste-generator, each process object is appended with a new property named "id", with a string value starting from 1.

findHotspots()

Parameters: -
Returns: -

This function goes through a copy of the array gatheredProcesses and identifies the hotspots according to the 80/20 rule, and pushes these hotspots into an array called identifiedHotspots.

First, gatheredProcesses is cloned, and a "percent" property is added to each object in it. This represents the waste proportion of this process with respect to the total amount of waste from all processes. The process objects are then sorted in ascending order based on their waste amount, and selection begins.

If only 1 process exists, it is automatically the hotspot. It is then pushed into identifiedHotspots and the function returns. Otherwise, we loop through the array and keep pushing the last process (the largest waste-generator) into identifiedHotspots until the waste gathered is 80% or more, as per the 80/20 rule.

If the waste is 80%, the hotspots are as identified and the loop breaks. Otherwise (more than 80%), a comparison is made between the current % and the previous value. Whichever % is closer to 80 will be chosen, along with the associated hotspots. If they are equally far away from, the current implementation will choose the lower value so that there will be 1 less hotspot for the user to prioritise.

At every iteration point of the loop, a check is done to see if the remaining (not yet chosen) processes generate equal amount of waste. In this event, the loop breaks and stops identifying new hotspots. This is because the selection is based on waste amount alone, and we do not have other information to choose between the tied processes.

Regardless of which of the above cases happens, a string of the results is set accordingly.

testForEqualPercent(array)

Parameters:

- array, an array containing process objects to be checked.

Returns: Boolean

This function goes through all process objects in the array parameter and checks if they share the same waste percentage. Returns a boolean representing whether they do. If the array contains less than 2 objects, returns false.

setResultString()

Parameters: -

Returns: -

This function stores the appropriate results string in the variable resultString, including the current gathered waste % rounded to 2 decimal places.

proceedWithHotspot(selectedTimeFrame, selectedYear, selectedMonth, normalise, callback)

Parameters:

- selectedTimeFrame, a string representing the data entry timeframe chosen by the user.
- selectedYear, an int representing the year selected by the user, if not the default value.
- selectedMonth, a string representing the month selected by the user, if not the default value. Contains name of month & year (year is required since e.g. May 2016 is not the same as May 2017).
- normalise, a boolean representing whether the user has chosen to normalise the waste amounts.
- callback, a passed function to be called. This is for working with asynchronous execution.

Returns: -

This function contains the set of functions for hotspot analysis that should only be called after the data entries have been retrieved from the database. The callback function is called at the end.

getEntriesForHotspots(selectedTimeFrame, selectedYear, selectedMonth, normalise, callback)

Parameters:

- selectedTimeFrame, a string representing the data entry timeframe chosen by the user.
- selectedYear, an int representing the year selected by the user, if not the default value.
- selectedMonth, a string representing the month selected by the user, if not the default value. Contains name of month & year (year is required since e.g. May 2016 is not the same as May 2017).
- normalise, a boolean representing whether the user has chosen to normalise the waste amounts.
- callback, a passed function to be called. This is for working with asynchronous execution.

Returns: -

This function is for fetching the relevant data entries from the database. It sends a GET request to getEntries.php, along with the timeframe information. When the response is received, the data entries are stored in the variable “entries”, and the next step of hotspot analysis is executed. If an error occurs, it is logged in the console & alerted to the user.

Exposed functions:

prepareHotspotData(selectedTimeFrame, selectedYear, selectedMonth, normalise, callback)

Parameters:

- selectedTimeFrame, a string representing the data entry timeframe chosen by the user.
- selectedYear, an int representing the year selected by the user, if not the default value.
- selectedMonth, a string representing the month selected by the user, if not the default value. Contains name of month & year (year is required since e.g. May 2016 is not the same as May 2017).
- normalise, a boolean representing whether the user has chosen to normalise the waste amounts.
- callback, a passed function to be called. This is for working with asynchronous execution.

Returns: -

This function is to be called by other components to initiate hotspot analysis process. It declares / resets the required variables and calls the next step of hotspot analysis.

getGatheredProcesses()
Parameters: -
Returns: gatheredProcesses, an array of the process compiled from the data entries

This function is to be called by other components to get the processes involved with the latest iteration of hotspot analysis. It is currently used for drawing of the Pareto chart.

getIdentifiedProcesses()
Parameters: -
Returns: identifiedProcesses, an array of the identified hotspots

This function is to be called by other components to get the identified hotspots. It is used to store the analysis results that will be used by the Measures Recommendation tool. The length of this array is also used to differentiate the Pareto chart colours.

getTotalWasteAmount()
Parameters: -
Returns: totalWasteAmount, a float representing the total waste generated by all processes in the latest iteration of hotspot analysis.

This function is to be called by other components to get the total waste amount. This is part of the results displayed to the user.

getResultString()
Parameters: -
Returns: resultString, a string containing the results of the latest iteration of hotspot analysis.

This function is to be called by other components to get the latest result strings, for displaying to the user.

getCurrentPercent()
Parameters: -
Returns: currentPercent, a float representing the proportion of the waste gathered from the hotspots in the latest iteration of hotspot analysis.

This function is to be called by other components to get the waste % gathered during the latest iteration of hotspot analysis. Used for drawing the dotted line in the Pareto chart.

services.js (dataService)

This section only applies to the service named "dataService" in the file. This service is meant sharing of data between different components, by first storing it here and then accessing this service from another component as required.

Note, this service contains a variable named phpServerName, which is used for accessing the php REST api. This is currently http://localhost/, which is for use with the Bitnami WAPP stack. If the php files are to be hosted elsewhere, the address only has to be changed here once, since the components that need to make send a request calls a function to get this variable.

Internal functions:

compileEvaluationResults(evaluationResults)

Parameters:

- evaluationResults, an array containing the evaluation entries fetched from the database

Returns: -

This function is to compile the evaluation result entries fetched from the database. There can be more than 1 entry pertaining to the same waste material (e.g. 1 avoidable cause & 1 diversion recommendation), and these entries will have to be combined.

There are 2 arrays in this function: wasteResults & addedWaste. wasteResults is for storing objects representing each unique waste, while addedWaste stores the names of the unique waste materials encountered (this is used for checking of new wastes & indexing).

The function loops through all entries in evaluationResults, and creates a new waste object in the wasteResults array if the waste has not been encountered before in the loop. The result contained in entry is then added into the appropriate waste object.

After the loop, the compiled results are then stored in a variable called evaluations, and the names of the wastes encountered stored in a variable called evaluatedWastes.

getThisWeekDates()

Parameters: -

Returns: sevenDates, an array containing 7 objects, each containing the date details of this week.

This function is for generating the dates for this week. Each week begins on a Monday and ends on a Sunday. The order of the objects in the array also begin from Monday. If today is not a Monday, the code goes back to the latest Monday and starts getting dates from there.

handleWeekResult(weekResults, thisWeekDates)

Parameters:

- weekResults, an array containing the waste data for this week that were fetched from the database.
- thisWeekDates, an array containing objects with dates of this week.

Returns: result, an array containing 7 objects, each containing dates of this week and their respective waste amounts.

This function is for compiling and returning the waste amounts from the database entries according to this week's dates. This is used for the dashboard's different views.

handleMonthResult(monthResults, thisYear)

Parameters:

- monthResults, an array containing the waste data for the months of this year that were fetched from the database.
- thisYear, an int for today's year.

Returns: result, an array containing 12 objects, one for each month of the year and their respective waste amounts.

This function is for compiling and returning the waste amounts from the database entries according to the months of the year. This is used for the dashboard's different views.

handleYearResult(yearResults)

Parameters:

- yearResults, an array containing the waste data across different years that were fetched from the database.

Returns: result, an array containing n objects, where n is the number of different years found in the database data entries.

This function is for compiling and returning the waste amounts from the database entries according to the years. This is used for the dashboard's different views.

handleProcessResult(results)

Parameters:

- results, an array containing the data entries fetched from the database.

Returns: result, an array containing objects for each unique process in the data entries, each containing information about the respective processes.

This function is for compiling and returning the process information from the database entries. Each object has the process name, total input, total output, total waste and efficiency percentage (output over input). This is used by the dashboard to display the top 3 (if any) inefficient processes over various views.

The information is compiled into objects in the result array, which are then sorted based on their efficiency percentage, in ascending order. The result array is then returned.

handleAverageCostsResult(averageCostsResult)

Parameters:

- averageCostsResult, an array containing the average costs of production & disposal that were fetched from the database (if any).

Returns: result, an object containing the average costs of production & disposal

This function checks and returns an object containing the average production and disposal cost. The passed parameter is the retrieved results from the database. If the results exist in the database, the object gains their respective values. If the results do not exist, this means these costs have not been initialised yet, and the object will gain a value of -1. This is used by the dashboard for the cost of waste calculation.

External Functions:

getPHPServerName()

Parameters: -

Returns: a string of the server name where the PHP files are hosted at.

This function is used by other components when they need to make a REST request. Note that this is only the name of the server's root, these other components still needs to provide the remainder of the file directory to their target PHP files.

hasEntries(callback)

Parameters:

- callback, a function to be called after the results of the GET request are received.

Returns: -

This function is for checking whether there are any data entries in the database. This is used by the dashboard to prompt users to upload data if there are none.

A GET request is made to the PHP server for the number of data entries. If some entries exist, the callback function is called with a boolean value of true passed in, otherwise, a boolean value of false is used.

setEvaluationResults(wasteObject, callback)

Parameters:

- wasteObject, an object containing the waste & its evaluation results.
- callback, a function to be called after the results of the POST request are received.

Returns: -

This function is for adding evaluation results into the database. The wasteObject parameter contains the waste name, and 4 arrays of strings, each string representing an evaluation result (if any). If the results were successfully added into the database, the callback function is called with a boolean value of true, otherwise a boolean value of false is used.

getEvaluationResults(callback)

Parameters:

- callback, a function to be called at the end of execution.

Returns: -

This function is for fetching the evaluation results stored in the database, if any. After the GET response arrives, the results are compiled and the callback function is called.

setProcessesAndWastes(data, numOfHotspots, isNormalised)

Parameters:

- data, an array containing information on the processes involved in hotspot analysis.
- numOfHotspots, an int of how many hotspots were identified.
- isNormalised, a boolean representing whether the hotspot analysis was conducted with the waste amounts normalised.

Returns: -

This function stores the passed parameters (hotspot analysis results) in this service.

getProcessesAndWastes()

Parameters: -

Returns: result, an object containing information of the last conducted hotspot analysis

This function compiles and returns information regarding the last conducted hotspot analysis. For each of the processes involved, an object is created to store its name, id, total waste amount and an array containing its associated waste objects. Each of the waste objects has properties for its name, amount, a boolean for whether this waste item has been evaluated previously, and the evaluation results if any.

The waste objects are initially created as unevaluated. The waste name is then checked with the arrays evaluatedWastes and evaluations to see if this waste item has been evaluated before. If so, the results are added into the waste object.

The results are placed into an object, containing the following:

- processes, an array for the process objects.
- numOfHotspots, an int for of how many hotspots were identified.
- isNormalised, a boolean representing whether the hotspot analysis was conducted with the waste amounts normalised.

This is then returned for use with Measures Recommendation.

getDashboardData(callback)

Parameters:

- callback, a function to be called at the end of this function

Returns: -

This function is used for providing the dashboard with the required data. It sends a POST request with dates information for fetching the information related to these dates. After the response is received, the results are processed using various internal functions, and are placed into an object which contains the following:

- hasEntry, a boolean representing whether the database contains any data entries.
- isInitialised, a boolean representing whether the database contains the matrix skeleton and process information.
- weekView, an array containing waste information for this week.
- monthView, an array containing waste information for the months of this year
- yearView, an array containing waste information for each year.
- todaydayProcesses, an array containing information on today's processes.
- monthProcesses, an array containing information on this month's processes.
- yearProcesses, an array containing information on this year's processes.
- averageCosts, an object containing information on the stored average costs.

The object is then passed into the callback function, to be used by the dashboard.

PHP Files

The documentation for this section is for the RESTful services in the folder "PHP Files/api" in the repository. For information regarding setting up the PHP environment and establishing a connection between the app and the database, refer to the "PHP Files/readme.docx".

db_connect.php

This file contains the database login credentials, and establishes a connection to it using the variable $conn.
$conn can then be used in other PHP files to perform queries on the database.
However, each of these other PHP files must include the following line:

```
require_once 'db_connect.php';
```

This line enables the file to access $conn, as it will now be in the global scope.

Note: the required line shown above is valid only for files in the same folder, and only if the file containing the credentials is named db_connect.php. If the files are not in the same folder, use the appropriate file path name.

addAverageCosts.php

This file is used by the controller for the initialisation page for saving the average costs of production & waste disposal. The costs are POSTed along with the request.
Old values in the database, if present, will be deleted first.

addEntries.php

This file is used by the controller for the data input page for inserting data entries parsed from an uploaded spreadsheet into the database (into the table BreadTalkData). It expects an array of the entries to be POSTed.
Each item in the array represents an entry to be added, containing values for year, month, day, role of the material (input, output or waste), process name, material name, amount, remarks (remarks currently used for identity of the data collector).

The table for data entries uses ID as its primary key. ID for a new entry is simply the largest existing ID in the table, incremented by 1. This is accounted for by first fetching the largest existing ID, and then incrementing it in the for loop in this file.

Each entry inserted into the table also has a value called uploaded. This value is a number representing the number of seconds since the Unix Epoch, which is initialised in this file under the variable $timestamp. This value is used to uniquely identify which entries were submitted in the same batch/spreadsheet.

All the SQL INSERT commands as performed as one transaction, so if for some reason any of them fails, none of them will be committed.

addEntriesAfterDeleting.php

This file is used by the controller for the data input page for inserting data entries parsed from an uploaded spreadsheet into the database. This file works the same way as addEntries.php above, except that the user has chosen to replace all entries that were uploaded in the previous spreadsheet batch.

Note: in the current implementation, entries that were added directly through the app dropdown list (not by browsing to a spreadsheet) will not be replaced. The uploaded timestamp for these entries are -1, whereas spreadsheet-uploaded entries will be a positive number. This is used to differentiate them.

The deletion occurs in the beginning of the transaction, and the current largest ID is retrieved after deletion. Finally, insertion of the entries occurs. Since this is all in a transaction, the previous batch of entries will be preserved if the transaction does not get committed.

addEntriesFromApp.php

This file is used by the controller for the data input page for inserting data entries directly through the app (by choose a process from the dropdown list, and inputting the weight for the associated materials).

This file works the same way as addEntries.php above, except that the the uploaded timestamp for these entries are -1, instead of a positive number. This is used to differentiate the method of entry upload. Additionally, the remarks column of these entries will have a value of “app”.

addEvaluationResults.php

This file is used to save the results of a waste material’s evaluations from Measures Recommendations.
It expects various information to be POSTed:

- Name of the waste material
- 4 arrays of strings, each containing evaluation results for avoidable causes, unavoidable causes, reduction recommendations & diversion recommendations. (note: the array(s) can be empty if the evaluation yielded no results)

All existing entries in the database with the same waste material name will first be deleted since we are going to replace with the new results.

Following that, the code loops through each of the 4 arrays, and for each result string, an entry is added into the EvaluationResults table.

All of the queries take place in one transaction.

addMatrixSkeletonAndProcesses.php

This file is used by the controller for the initialisation page for saving information regarding the user’s matrix skeleton and process information. It expects 2 arrays to be POSTed, one containing data on the skeleton, the other for process information.

All existing entries in the database pertaining to the matrix skeleton and process information are first deleted (from tables MatrixSkeleton & ProcessInfo). Next, the code loops through both arrays and adds each piece of information into the respective tables.

Finally, all existing entries pertaining to waste evaluation results are deleted (from the table EvaluationResults). This is because now that a new series of process information has been uploaded, the waste materials in the new system may now be different from the ones in the existing results, rendering the stored results useless. Additionally, the old results may share a same waste name with a waste in the new system, even if they are not the same thing (and thus should not share the same results).

All of the queries take place in one transaction.

getDashboardData.php

This file is used by the dataService in services.js for fetching and compiling the information required by the dashboard.

It expects the following data to be POSTed:

- weekDates = an array of objects containing dates for this week (starts on Monday).
- month = number representing this month (note: JavaScript months start from 0, so take note of subtracting 1, eg April is 3. This is also reflected in the data entries.).
- year = number representing this year.
- today = number representing day of today

It performs the following queries:

- Fetch number of entries in BreadTalkData table. This is required by dashboard to inform user if there are no data.
- Fetch number of entries in MatrixSkeleton table, as well as ProcessInfo table (separately). This is required by dashboard to inform user that initialisation has not been done.
- Fetch waste data for each day of the current week, each month of the current year, and each year (separately). These are required by the dashboard for the different timeframe views.
- Fetch process data for today, this month, and this year (separately). This is required by dashboard for showing the top 3 inefficient processes in the respective timeframe views.

The results of the queries are added into an array, which is finally returned.

getEntries.php

This file is used by hotspot service and normalisation controller to fetch the data entries from BreadTalkData table, according to the specified timeframe.

It expects the following data to be sent along with the GET request:

- timeframe = a string of the type of timeframe chosen by the user
- year & month = numbers representing the user-chosen year & month for the timeframe

The query result is then returned.

getEvaluationResults.php

This file is used to fetch all entries from EvaluationResults table, if any, and return it. Required upon loading of Measures Recommendation.

getProcesses.php

This file is used by the controller for data input, in order to populate the dropdown list for adding entries directly via the app.

getSkeleton.php

This file is used by the controller for normalisation, in order to retrieve and build the matrix skeleton.

getTimeframe.php

This file is used by the controllers for hotspot & normalisation. It fetches all distinct combinations of year and month values that exist among the data entries in the database. This is for determining which timeframe options are available for the user.

Database Tables

Note: In the Amazon RDS server, a database instance named “FLW_APP” was created and being used. This name is part of the information required for the PHP/PDO to establish a connection to the database. So if a different instance is used, the change has to be reflected in db_connect.php.

The following tables are all in the FLW_APP database instance.

AverageProductionAndDisposalCosts

Columns:
- ProductionCost = float, average cost for 1 kg of production.
- DisposalCost = float, average cost for 1 kg of waste disposal.

Primary Key: ProductionCost & DisposalCost

BreadTalkData

Stores the individual data entries regarding production.

Columns:
- ID = int, a number to differentiate each entry.
- Year = int, the year for the data recorded by the entry.
- Month = int, the month for the data recorded by the entry (note: JavaScript months start from 0, so April is 3. This applies here as well).
- Day = int, the day for the data recorded by the entry.
- InOutWaste = varchar(6), a string representing the role (input / output / waste) of the material recorded by the entry.

- Process = varchar(100), name of the process for the data recorded by the entry.
- Material = varchar(100), name of the material for the data recorded by the entry.
- Amount = float, the amount for the material recorded by the entry.
- Remarks = varchar(100), a string for any other information, currently used to record name of the person who collected the data (if the entry was added via spreadsheet. If the entry was added through the app's dropdown list, a value of "app" is used instead.
- Uploaded = int, a timestamp representing the number of seconds since the Unix Epoch. This value is used to uniquely identify which entries were submitted in the same batch/spreadsheet. If the entry was added through the app's dropdown list, a value of -1 is used instead of a positive number.

Primary Key: ID

EvaluationResults

Stores the results generated by Measures Recommendation.

Columns:
- Waste = varchar(100), the name of the waste for this entry.
- Type = varchar(20), the type of the result, whether it is for avoidable or unavoidable cause, diversion or reduction. (currently, the strings used are "avoidable", "unavoidable", "diversion", "reduction". The JavaScript code looks for these values).
- Value = varchar(100), a string representing the analysis result.

From this format, let's say the user saves the evaluation results for 1 waste material. In the results, 1 avoidable cause was identified, and 2 diversion methods were recommended. The will result in the insertion of 3 entries.

Primary Key: Waste, Type & Value

MatrixSkeleton

Stores the cell information of the uploaded matrix skeleton. The matrix includes all processes & materials in the system. All numeric cells will have a value of 0 (will only be filled when user performs normalisation) except for static processes inserted for intermediary purposes.

Columns:
- Row = int, a number representing the row index of this entry. Starts from 0.
- Col = int, a number representing the column index of this entry. Starts from 0.
- Value = varchar(50), the value of the cell represented by this entry. It may contain headers, process names, or numeric values, depending on which cell this is.

Primary Key: Row & Col

ProcessInfo

Stores the information of the uploaded processes. This includes all processes in the system, and every material associated with each process.

Columns:
- Process = varchar(50), process name for this entry.
- Component = varchar(50), material name for this entry.
- Role = varchar(6), represents what is the role (inout/out/waste) of this entry's material within this entry's process.

From this format, suppose there is a process called x, containing materials y and z as input and output respectively. This will be represented by 2 entries, both sharing the same Process name x. One entry will have Component name of y and Role value of input, while the other will be z and output.

Primary Key: Process, Component & Role