1. USER INPUTS DATA

   The trigger for the search component is when the user clicks on one of the two search buttons.

   - Search by keyword will trigger **/searchManualKeyword**
   - Search by UPC will trigger **/searchManualUPC**

   The value to pass to the server side for the research will be the string entered by the user in the appropriate input field.

2. CLIENT COMMUNICATES WITH SERVER

   Again, depending on the button chosen by the user, the client side will communicate the input value and the trigger verb to the client side (Node.js).

/searchManualKeyword will access the handleGetProfileByKeyword() function - Since the keyword is already the desired format for the next function, we can only store that brand name in the new **Item** object (newSearchedItem.brand).	**/searchManualUPC** will access the handleGetProfileByUPC() function - From the UPC, we need to find the new Item's name, brand and image URL. In the future, this **Item** object will also be pushed in a database so users can view their search history. To find these three Item components, we call the getUPCPage(upc) function, which makes a 'POST' request to the UPCitemDB API. If the UPC is conform and found in their database, we can plug the information needed in our newSearchedItem Item object (newSearchedItem.name, newSearchedItem.brand, newSearchedItem.imageURL). We

	also define a boolean status variable (foundBrandbyUPC) to be true. This allows the handleGetProfileByUPC() to continue to the next steps in its function.

3. FIND ITEM AND COMPANY INFORMATION ON THE SERVER SIDE

When the main functions (handleGetProfileByKeyword() or handleGetProfileByUPC()) have defined the newSearchedItem.brand variable, they call the handleGetProfile() function, which will handle the information retrieval to define all the components of our new **Company** object.

a) Firstly, the function calls the getMotherCompany(brand) async function, which is a 'GET' request to the Wiki API. The function uses both the *request* and the *wikiapi* modules. The function resolves a string of the result of the request after Regex manipulations. Also, the function defines different status boolean variables:
   i) If the request with the brand received by the client allows us to find the <currentowner> or <owner> of the brand from our request, the variable foundCoFromBrand is true. The owner value is then resolved as the result string.
   ii) If the request with the brand received by the client does not allow us to find the <currentowner> or <owner> of the brand but allows us to find a <manufacturer>, the variable foundCoFromBrand is false, while manufacturerAvailable is true. The manufacturer value is then resolved as the result string.
   iii) If the request with the brand received by the client does not allow us to find the <currentowner>, the <owner> or the <manufacturer>, both the variables foundCoFromBrand and manufacturerAvailable stay false.

When the function is resolved, the handleGetProfile() function verifies the status of the result. If the variable foundCoFromBrand is true, the value resolved defines the newSearchedCompany.name. Else if the foundCoFromBrand is false but manufacturerAvailable is true, the function will call again the getMotherCompany(manufacturer) with the manufacturer as the searched keyword. If both the variables foundCoFromBrand and manufacturerAvailable are false, the boolean abortMission is true. This variable will forbid the next functions to be called without a defined company name and help status communication with the client in the future.

b) If abortMission is not true, handleGetProfile() calls the getMarketSymbol(company) function which the result will define the newSearchedCompany.financials.symbol. The financials variable of the **Company** constructor is also an object; the **FinancialProfile** class. This function has a time-bounded promise for a 'GET' request to the YHFinance API (distributed by RapidAPI). From a company name, the request finds the different profiles associated in the Yahoo Finance database. If the result is not undefined, the function resolves the market symbol of the first result entry (JSON.parsed), which is the most probable one.

c) From that market symbol, we can obtain the profile of the new searched company. If abortMission is not true, handleGetProfile() calls the getYHFinanceProfile(marketSymbol) which is void. This function has a time-bounded promise for a 'GET' request to the YHFinance API (distributed by RapidAPI). From the market symbol, the request finds the different information included in the 'Profile' section of a company on Yahoo Finance. In our case, we use the parsed result of the request to define the profile variable of the **Company** constructor is also an object; the **CompanyOverview** class. With a defined result, this function defines:

   i) newSearchedCompany.profile.sector,
   ii) newSearchedCompany.profile.industy,
   iii) newSearchedCompany.profile.employeeNb,

    iv) newSearchedCompany.profile.adress (the three entries).
    v) newSeachedCompany.profile.companyOfficers (all the company officers and their position, and their salary if available)

d) Again from the market symbol, we can obtain the financials of the new searched company. If abortMission is not true, handleGetProfile() calls the getYHFinanceFinancials(marketSymbol) which is void. This function has a time-bounded promise for a 'GET' request to the YHFinance API (distributed by RapidAPI). From the market symbol, the request finds the different information included in the other financial sections on Yahoo Finance. With a defined result, this function defines:
    i) newSearchedCompany.financials.grossRevenue
    ii) newSearchedCompany.financials.grossProfit

   This function then defines the newSearchedCompany.financials.profitMargin from the getProfitMargin() function in the FinancialProfile class.

e) Then, from the same market symbol, the handleGetProfile() function calls the getPayrollRatio(markerSymbol) which resolves also void. This function uses the mongoose module, and the MedianWages Schema defining the MEDIANWAGES collection in my CheckoutApp MongoDB database. This collection is built on a .xml file that I created myself by copying the data from the *Executive Paywatch* from the AFL-CIO (2021). With a defined result, this function defines:
    i) newSearchedCompany.financials.medianWorkerPayroll
    ii) newSearchedCompany.financials.payrollRatio
    iii) newSearchedCompany.financials.fiscalYear

f) Finally, from the company name, the handleGetProfile() function calls the getFoodAndAgricultureBenchmark(companyName) which resolves also void. This async function uses the mongoose module, and the FAABSchema defining the FAABRESULTS collection in my CheckoutApp MongoDB database. This collection was available on a .csv file that I exported in .xml format from the *Food and Agriculture Benchmark* from the World Benchmarking Alliance (2021) website. To interpret the different results, I also created myself a FAABLEGENDS collection with the description of the benchmarks of the WBA for this study, also available on their website as a PDF document. This legend can be accessed through calling the getLegend(code, q). Depending on the query the legend returns a long string of describing the code/result. With a defined result, this function defines:

   i) newSearchedCompany.workersSocialInclusion[0]

      1) Index [0] because it is our first source in this category - one **Blurb** object per index
      2) For each subcategory/aspects of this main category, a **subBlurb** object is pushed in the **Blurb** index[0].subBlurbs array (newSearchedCompany.workersSocialInclusion[0].subBlurbs);
      3) The information about the specific research are also plugged (newSearchedCompany.workersSocialInclusion[0].id, newSearchedCompany.workersSocialInclusion[0].subject, newSearchedCompany.workersSocialInclusion[0].rating, newSearchedCompany.workersSocialInclusion[0].assessmentYear, newSearchedCompany.workersSocialInclusion[0].source, newSearchedCompany.workersSocialInclusion[0].nextAssessment, newSearchedCompany.workersSocialInclusion[0].description)

4. SEND BACK THE RESULTS TO THE CLIENT

When the handleGetProfile() async function is over, we are back either in the handleGetProfileByKeyword() or the handleGetProfileByUPC() main async functions. With the obtained results, we create a temporary object **scannedResult** to pass to the client with the response.send(scannedResult). This object contains the values of abortMission, newSearchedItem and newSearchedCompany.

5. DISPLAY THE RESULT ON THE CLIENT SIDE FOR THE USER

If the abortMission variable received back by the client is true, the function displayErrorMessage() will indicate to the user that no result was found from their data input. If it is false, then the function will call displayItem() and displayCompany() which takes care of displaying the found information in the right sections of the page. Some manipulations are done for some strings to append (separate) each list item in separate <li> in the html - see verifyListDisplay(). Other verifications are done like in the function verifyLink(), which removes the empty links for external resources. Finally, there are also some event handlers for the company resources tabs to be collapsible at different levels so the user can easily go deeper in one subject at a time.