Project Final Prototype Report

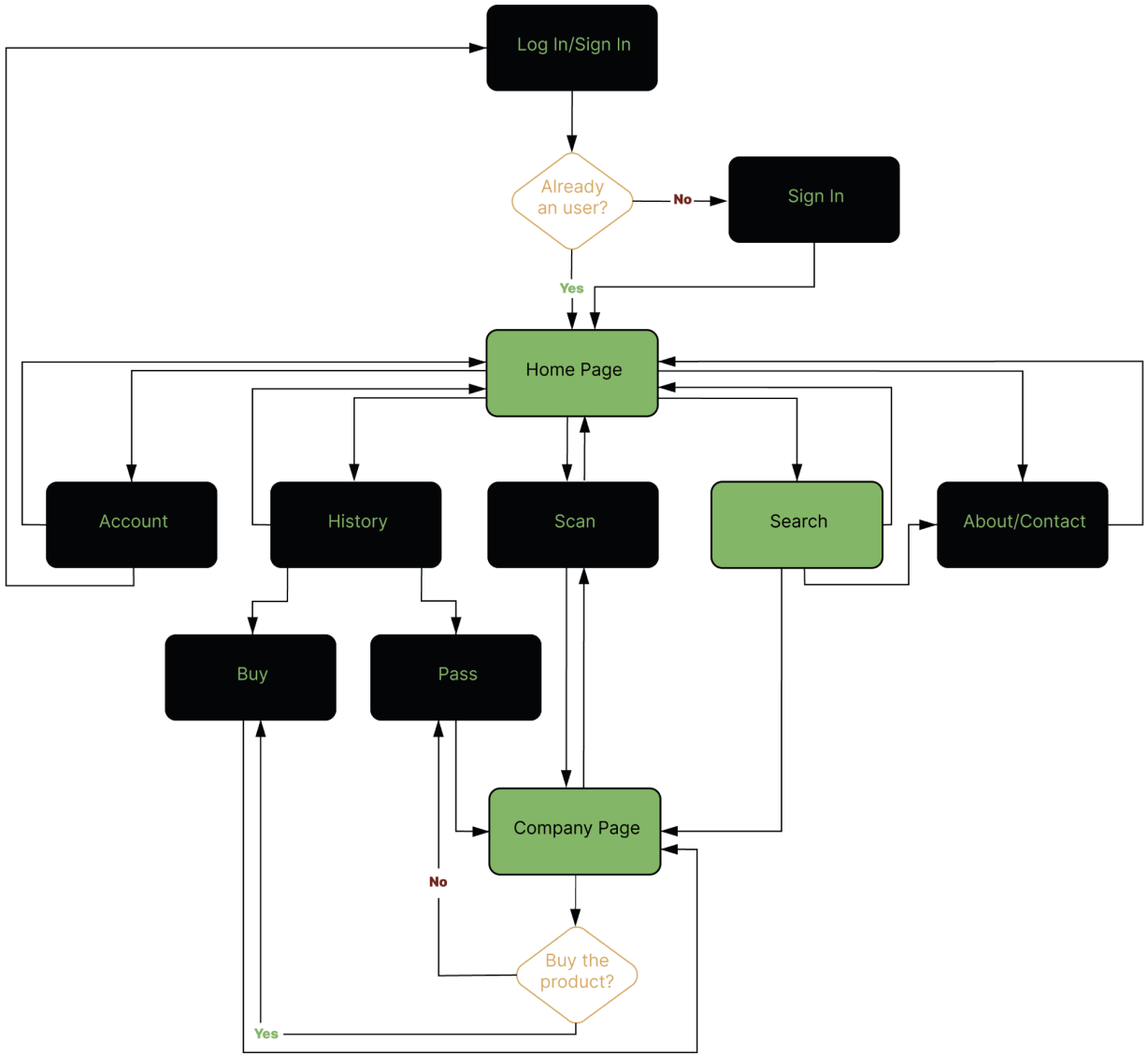
CART 451, Fall 2022

By Justine Lardeux

HIGH-LEVEL DESCRIPTION

This project is an app that allows good consumers to get a fuller and quicker picture of the different companies selling food across america. Everytime an user scans a product's UPC code (or enters it in the search bar), the program searches into different databases to identify the mother company of the brand and different outsourced assessments regarding their workers' treatment, its management governance, its environmental impacts, and its nutritional values. After displaying the company's page, the user then indicates if he decided to buy or not the product and the reason(s) behind it. This allows the users to revisit their different buying decisions with a timestamp and visual depictions of their reasons.

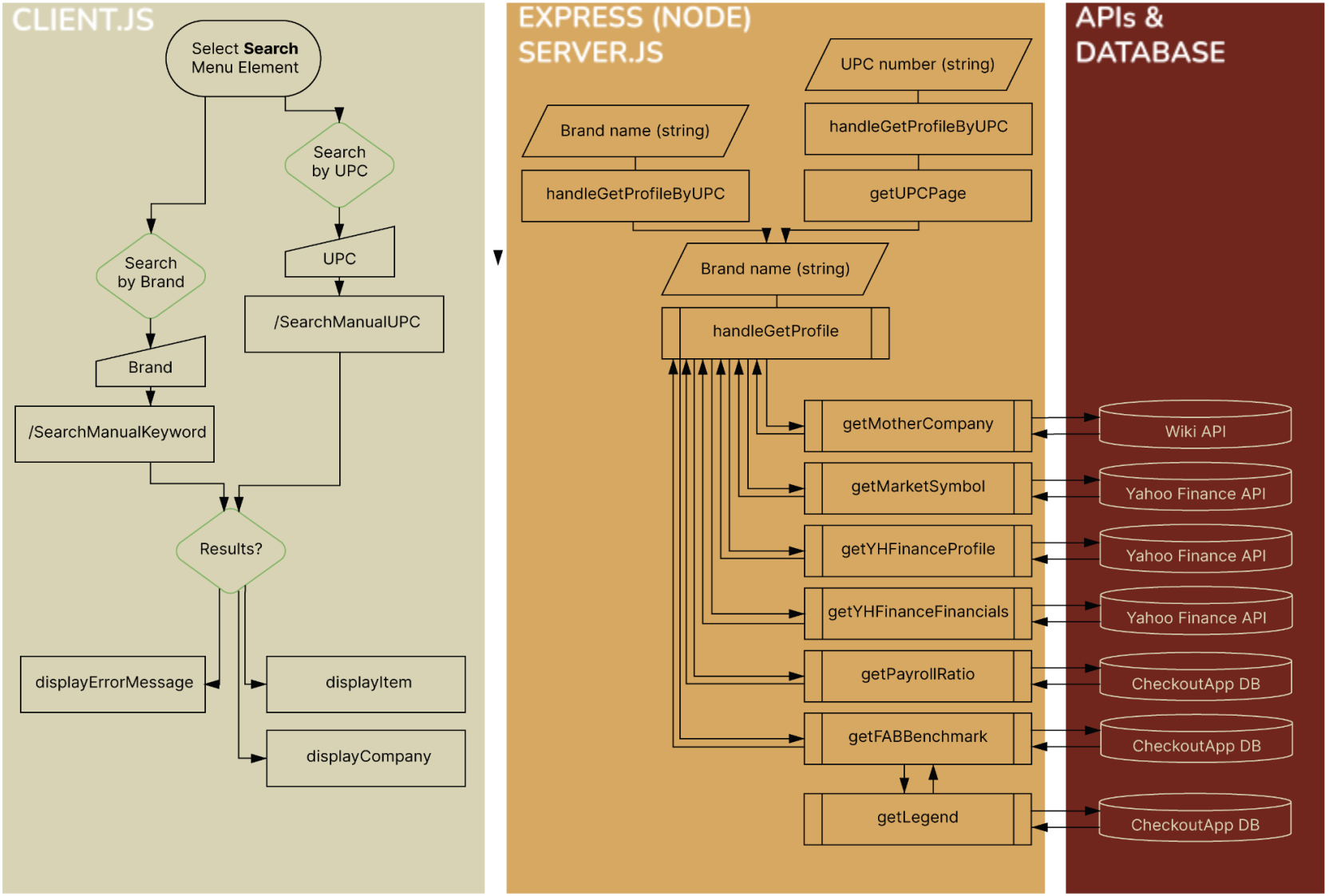
GUI NAVIGATION FLOWCHART



For this final class submission, I had to reduce the scope of the project to be able to have a working and clear project piece.

So, I focused the project to the **search** component of the initial concept. This part is the most crucial one, since it handles the research through different databases to obtain a company profile on different predetermined aspects. You can view the isolated working components in green in the GUI navigation flowchart

TECHNOLOGIES IMPLEMENTED SO FAR



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.

TECHNOLOGIES TO IMPLEMENT IN THE FUTURE

SEARCH COMPONENT

Being the component of the project I decided to implement within the frame of this class, it is much more clear what needs to be implemented or reworked. On the server side, I would like to work more on the search wikiAPI search so I can find more results. As of now, the system does not try to find the company directly from the keywords, it can only consider it as a brand (not the main company). Also, if a keyword has some special characters, the system will majoritarily not find any result. Working on this request would greatly improve the efficiency of my project. On the client side, I would like to work on the design, such as prioritizing the information so it is easier to read, while being careful to not prioritize biased elements. Also, the interactive components are currently elementary. I still need to add a loading page to indicate the user that the server side is looking for their search prompt.

HISTORY COMPONENT

On the search page, when a company is found, the user is prompted to choose between the 'BUY' and the 'PASSED' option. As of now, these choices are not active, but I would like to be able to push into a database the different products a user has searched/scanned and their decision to buy it or not. I would also like to implement different reasons the user can select to explain their decision. The userId, their choice, the reasons and the **Item** object will then be pushed into a database for me to be able to make assessment on the collected data in the future. It will also allow the users to view their history and make their own observations.

LOG IN/SIGN IN/ACCOUNT COMPONENTS

To have a functioning history, these three components need to be implemented. The user needs to be able to create an account (username, password and city only). This will allow me to allocate each search to each user for their history, and to also collect demographic data. This information can be modified through the 'account' section in time. The account will also be deletable. The

password encryption will be done with Bcrypt, a password hashing function compatible with Node.js. The accounts data will also be located in their own collection on my CheckoutApp MongoDB database.

SCAN COMPONENT

Since the app is meant to be used on the go, at the grocery store, it will be important to implement the scanner option. I will continue exploring Quagga.js to have a free option. However, I will monitor the new updates and products that could help me achieve this component on all devices (not just Android devices). Ultimately, I only need a UPC code reader, since the manipulations with the UPC are already written from the Search section.

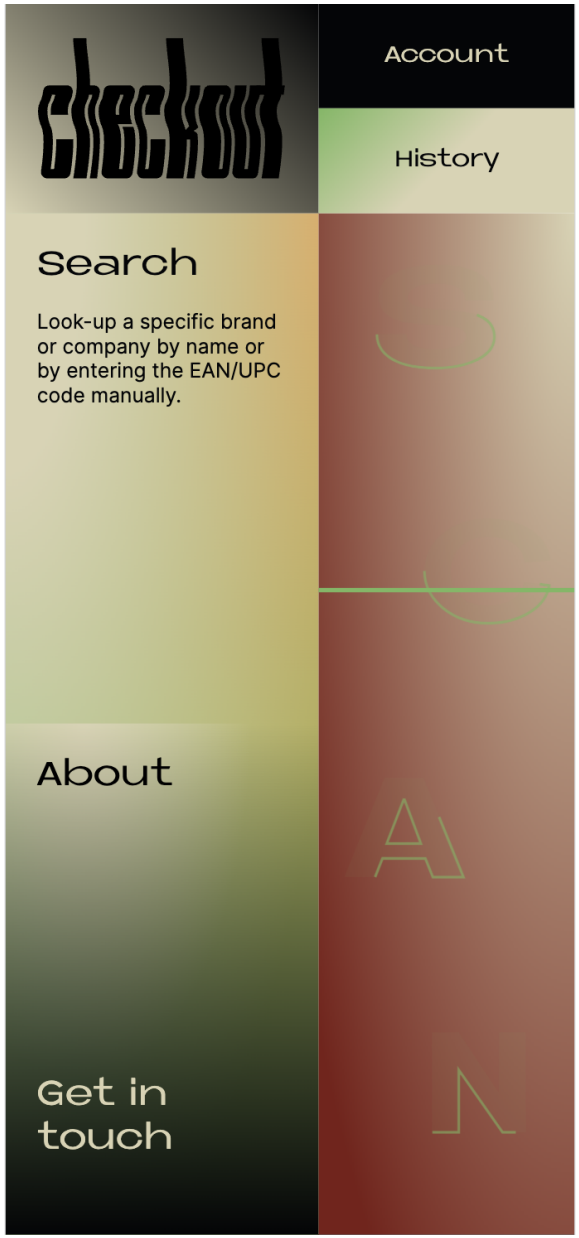
ABOUT/CONTACT COMPONENT

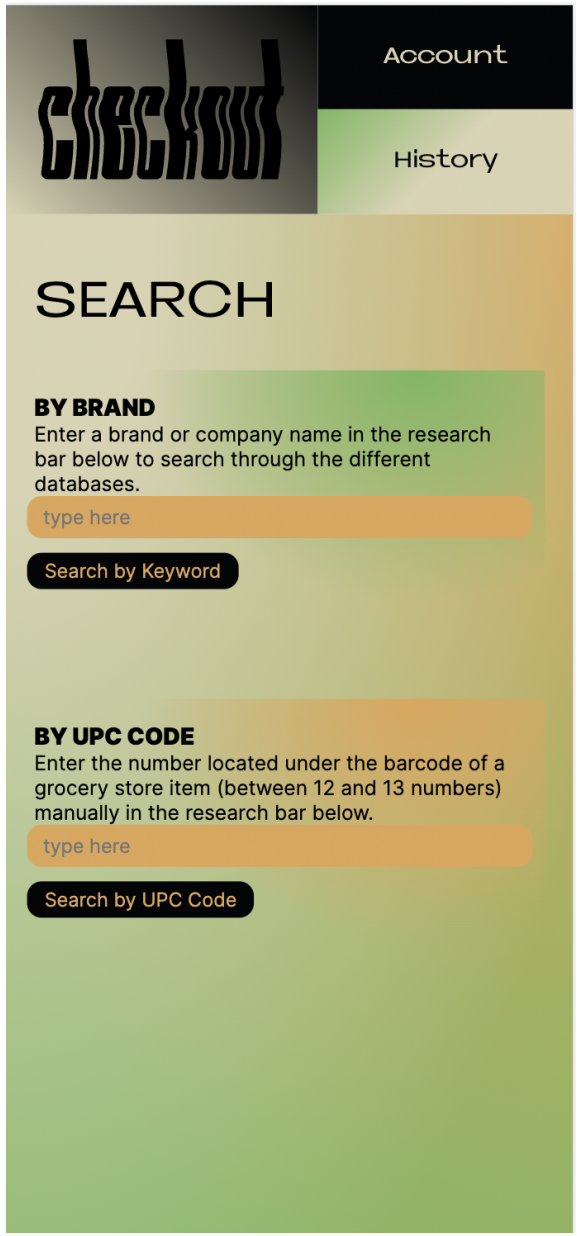
The about page will be static, but will contain all the sources and information about the project. The contact section will be situated just below and will be an hugely important component to widen my database. As of now, the brands that can be found are big ones, and are mainly American. I would like to be able to find more variety of brands, especially living in Canada. With the contact section, I want to allow users to request to add in the UPC database. The UPCitemDB allows keyholders to push in their API database directly.

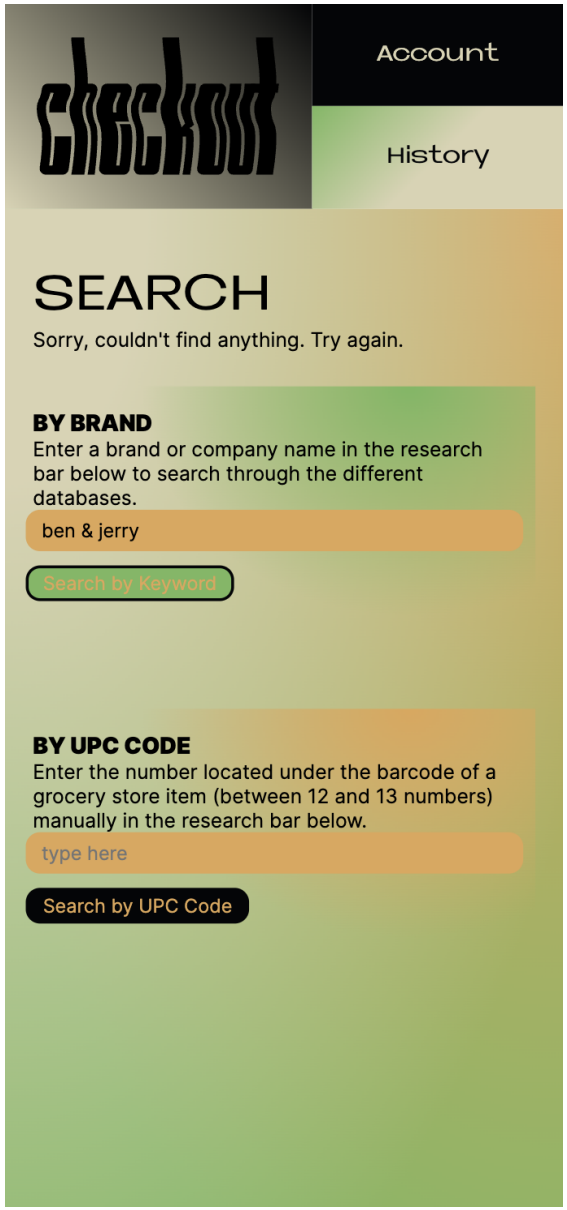
CONCLUSION

Ultimately, the search component implemented through this class is a good representation of my intended proposal. I was able to achieve all the research and data handling for the company profile, financials and external sources about its habits. The other components have yet to be developed, but the main part of the project is functioning accordingly. A lot of implementation still needs to be made to have more success in the searches and some better data representation could be explored. The project needs to eventually become an app able to scan for it to satisfy the purpose of the project.

SNAPSHOTS


checkout
Account
History
Search
Look-up a specific brand or company by name or by entering the EAN/UPC code manually.
About
Get in touch


checkout
Account
History
SEARCH
BY BRAND
Enter a brand or company name in the research bar below to search through the different databases.
type here
Search by Keyword
BY UPC CODE
Enter the number located under the barcode of a grocery store item (between 12 and 13 numbers) manually in the research bar below.
type here
Search by UPC Code


checkout
Account
History
SEARCH
Sorry, couldn't find anything. Try again.
BY BRAND
Enter a brand or company name in the research bar below to search through the different databases.
ben & jerry
Search by Keyword
BY UPC CODE
Enter the number located under the barcode of a grocery store item (between 12 and 13 numbers) manually in the research bar below.
type here
Search by UPC Code

```
setup_query_modules: 3 modules de requête trouvés : prop, list, meta
wiki_API_page: N'existe pas: [[Ben & jerry]]
Infobox is undefined for this page - cannot retrieve information
data is in main loop: undefined
foundCoFromBrand= false
manufacturerAvailable= false
NO LUCK HERE
back in the main function to send the info to the client
```

```
v49u37r01de@MacBook-Pro-de-Justine Locusts % node server.js
listening on port:: 4200
are here
UPC received from client: 052000039689
Product name: ORANGE THIRST QUENCHER POWDER, ORANGE
Brand: Gatorade
ImageURL: http://scene7.samsclub.com/is/image/samsclub/0005200003968_A?$img_size_211x208$
find company information from: Gatorade
get_API_parameters: Mise en cache les informations  concernant les modules API de enwiki : chemin du mo
dule=query+siteinfo
get_API_parameters: Mise en cache les informations  concernant les modules API de enwiki : chemin du mo
dule=query
setup_query_modules: 3 modules de requête trouvés : prop, list, meta
infobox is defined
company found directly from the brand
data passed: PepsiCo
data is in main loop: PepsiCo
foundCoFromBrand= true
manufacturerAvailable= false
company found from brand :)
Searching market symbol of: PepsiCo
db opened
retrieving company habits information from the databases
back in the main function to send the info to the client
```

SCANNED

ORANGE THIRST QUENCHER POWDER, ORANGE

Gatorade

PepsiCo

BUY **PASS**

PepsiCo

Sector: Consumer Defensive
Industry: Beverages—Non-Alcoholic
Approximately 309000 full time employees.

700 Anderson Hill Road
Adress Line 2
United States

FINANCIAL PROFILE

2021/TTM

Gross Revenue: 83.64B
Gross Profit: 44.36B

Profit margin: 53.04%

Median Annual Salary: $52,297
The CEO makes 488:1 times more than the median employee salary.

Key Executives, annual salary

Mr. Ramon Luis Laguarta, Chairman & CEO, 12.35M
Mr. Hugh F. Johnston, Vice Chairman, Exec. VP & CFO, 6.73M
Mr. Kirk Tanner, Chief Exec. Officer of PepsiCo Beverages North America, 4.82M

Gross Revenue: 83.64B
Gross Profit: 44.36B

Profit margin: 53.04%

Median Annual Salary: $52,297
The CEO makes 488:1 times more than the median employee salary.

Key Executives, annual salary

Mr. Ramon Luis Laguarta, Chairman & CEO, 12.35M
Mr. Hugh F. Johnston, Vice Chairman, Exec. VP & CFO, 6.73M
Mr. Kirk Tanner, Chief Exec. Officer of PepsiCo Beverages North America, 4.82M
Mr. Silviu Yeugeniu Popovici, Chief Exec. Officer of Europe, 6.23M
Mr. Steven C. Williams, Chief Exec. Officer of PepsiCo Foods North America, 2.75M
Mr. Gregg Roden, Exec. VP & COO
Mr. Rene Lammers, Exec. VP & Chief Science Officer
Mr. Seth Cohen, Sr. VP & Global Chief Information Officer
Mr. David J. Flavell, Exec. VP, Gen. Counsel & Corp. Sec.
Ms. Jane Caroline Wakely, Exec. VP, Chief Consumer & Marketing Officer and Chief Growth Officer of International Foods

GOVERNANCE AND MANAGEMENT +

WORKERS AND SOCIAL

ENVIRONMENT

NUTRITION

GOVERNANCE AND MANAGEMENT +

World Benchmarking Alliance 2021

Food and Agriculture Benchmark

Governance and strategy + 9,2/10

Sustainable development strategy + 2/2

Governance and accountability for sustainable development + 1,5/2

Pros:

The company:

- **Discloses having persons, teams or committees within the company who are responsible for the implementation of its sustainable development strategy.**
- **Discloses that accountability and oversight for its sustainability strategy lies with the highest governance body.**
- **Discloses its decision-making process and oversight responsibilities, as well as its management/executive-level OR highest governance body remuneration policy regarding sustainability.**

Cons:

The company does not disclose the management/executive-level AND highest governance body remuneration policy regarding sustainability, covering sustainability topics across all three benchmark measurement areas (environment, nutrition and social inclusion).

See: undefined

Stakeholder engagement + 2/2