BOOSE MUSE: A COCKTAIL KNOWLEDGE BASE AND BAR ADMINISTRATION TOOL

A REPORT SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF BACHELOR OF SCIENCE
IN THE FACULTY OF SCIENCE AND ENGINEERING

2023

Samuel Avery Zureick

Project Supervisor: Dr. U. Sattler
Department of Computer Science

Contents

Word Count: 12337

Abstract

The Booze Muse project is aimed at creating a tool for bar administrators and cocktail consumers that allows for the flexible querying of cocktails and their ingredients with very little technical knowledge. The tool includes all of the necessary information for a user to craft a myriad of cocktails from start to finish, including their ingredients, preparation method, glassware, and garnish. The cocktails and their components are stored as an ontology, along with allergen information. Ingredients and allergens are populated into the ontology by hand, while all of the cocktail information is scraped from the trusted cocktail archive 'Punch Drink' and inserted into the ontology in an automated fashion, allowing for the rapid and accurate development of an extensive cocktail knowledge base. The graphical user interface allows users to browse through cocktails stored in the ontology, perform advanced queries involving the inclusion or omission of allergens or ingredients, automatically generate and export balanced, stylized menus along with their accompanying allergy information, and filter through cocktails through allergen exclusion. The results of quantitative testing show poor scalability and maintainability, while the usability study highlights the intuitive design of the interface and the real-world usefulness of the application. In the conclusions, methods of increasing scalability, maintainability, and usefulness are highlighted.

Declaration

No portion of the work referred to in this report has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

I would like to thank Uli Sattler for her continued help and support throughout my time at the University of Manchester. Additionally, I would like to thank David Cole for his help in developing my knowledge. Finally, I would like to thank Aaron and Andrea Zureick for their love and support throughout my whole life.

Acronyms and Abbreviations

Acronyms and Abbreviations that appear throughout:

SPARQL SPARQL Protocol and RDF Query Language

OWL Web Ontology Language

RDF Resource Description Framework

GUI Graphical User Interface

UI User Interface

XML Extensible Markup Language

HTML Hypertest Markup Language

CSS Cascading Style Sheets

API Application Programming Interface

OOP Object-Oriented Programming

List of Figures

Chapter 1

Introduction

1.1 Project Description

This project is focused on crafting a bar administration tool driven by an intuitive user interface where the underlying representation of the drinks and ingredients takes the form of an ontology. The tool will contain an expansive and relevant knowledge base of cocktails, stored alongside their ingredients and other information such as preparation instructions, proper glassware, and garnish. Managing a high-end bar or restaurant takes an extreme amount of administrative overhead, and often the creativity of the institution and its staff is obstructed by the essential elements of service- dedicating time and resources towards creative efforts is an easy area to cut corners when delegating tasks, and this ends up detracting from the overall success and value of the business. The Booze Muse tool aims to simplify this process by allowing bar administrators to fine-tune their guests' experience without having to search through page upon page of cocktail archives. In a bar or restaurant, managing food allergies correctly and safely is critical to the safety of the customers and the integrity of the business. This tool also aims to simplify the allergen handling process by automatically classifying cocktails as containing allergens and exporting this information in a way that is easy to understand for both employees and customers. The cocktail knowledge base is automatically populated in order to maximize the number of cocktails stored in the ontology with a high degree of robustness and maintainability.

1.2 Functional Requirements

The goals and requirements set out for this project are:

1. Design and implement an ontology representing the cocktail domain which is capable of:

   (a) Accurately describing individual cocktails in terms of ingredients, preparations, garnishes, and serving style.

   (b) Representing a wide array of individual cocktails by including an extensive ingredients list.

   (c) Reliably representing the major allergens contained within the individual ingredients.

   (d) Maintainability through the introduction of new drinks and ingredients.

2. Design a web scraping tool that is able to:

   (a) Scrape drinks from trusted web sources.

   (b) Accurately convert these drinks from text data to a form that can be stored in an ontology.

   (c) Appropriately handle uncertainty during the conversion process.

3. Design a user interface that allows the user to:

   (a) Easily interact with the contents of the ontology with complete abstraction from the underlying data structures and no prior exposure.

   (b) Search through all cocktails in the ontology, and provide for each cocktail an ingredients specification, method for creation, and list of major allergens.

   (c) Search for subsets of cocktails in the ontology that omit user-specified allergens.

   (d) Generate and balanced menus that conform to high-end bar standards.

   (e) Search for cocktails given an arbitrary level of specification relative to ingredients and allergens to include or omit.

   (f) Export custom menus, along with health and safety complicit allergen documentation.

1.3 Evaluation Strategy

As the application's target audience is bar administrators and those interested in cocktails, it should not be expected that the user will have a high degree of technical expertise. Therefore, a usability test will be one component of the overall evaluation. Users with no in-depth technical knowledge or knowledge of ontologies should be able to easily use even the more advanced features of the application with no coaching. The users will be given sequences of tasks to complete, followed by feedback questions to assess how intuitive the interface is and where it can be improved, as well as expose any confusing or frustrating aspects. This will be a thinking-aloud usability study, where the users will be encouraged to speak aloud during the whole test and express their thoughts throughout. In addition to the usability study, there will be some manual and automated testing to evaluate the technical performance of the system. In order to evaluate the web-scraping process, the classification accuracy of ingredients will be evaluated, and instances of misclassification that could be potentially disastrous (e.g. alcohol in a non-alcoholic cocktail, missed allergen classification, etc) will be noted. The scalability of the application will be tested using an automated method, where the time to perform specific queries will be observed given changing numbers of ingredients and cocktails. The robustness of the application will be observed through the trial inclusion of new cocktails and ingredients.

1.4 Report Structure

This report first introduces existing systems and research done in regard to food and drink ontologies, ontologies, and other systems for managing allergies, and the hierarchical structure of drinks. Then, the overall system architecture is introduced, along with testing and evaluation methodology. Following this are the results of the usability and technical testing, along with an evaluation of these results. Finally, further areas of improvement and conclusions are introduced.

Chapter 2

Background

2.1 Overview

The restaurant industry recently has had to start to introduce much more technology into their workflow than ever before. The COVID-19 pandemic forced many restaurants to introduce online booking systems, online menus, self-service kiosks, delivery service compliance, and much more, all of which lie atop a robust technical infrastructure. Additionally, the pandemic has put new stress on the health and safety standards for restaurants, with consumers now much more knowledgeable and concerned about this area of the industry than they were before the pandemic [13].
With this new technological backbone comes the opportunity for businesses wishing to excel in their craft to develop and use tools not just for compliance, but to improve their efficiency and the customer experience. Menus are already present in some online forms for business-to-consumer interactions, and back-of-house staff often has a range of systems in place for consolidating and managing menus, dish specifications, stock levels, ordering, and many more administrative responsibilities. A unified solution, in which there is one digitized knowledge base that encompasses everything from large menus to the nutritional information of individual ingredients, would provide the consumer with a much broader understanding of the food and drink that they are enjoying and would allow the consumer to make better choices for themselves given their dietary requirements and special requests. Additionally, this could increase administrative efficiency by allowing for the quick composition of menus, simplifying changes and modifications to dishes, cocktails, and ingredients by automatic propagation through the restaurant infrastructure, and allowing bartenders and chefs to quickly and easily source recipes that conform to the exact specification of the guest.

2.2 Technical Terminology

The knowledge about cocktails and ingredients for this application will be stored in an **ontology**, which is a collection of (typically) classes, attributes, and the relationships between these.

One way of developing and formatting ontologies is through the **OWL 2 Web Ontology Language** (OWL2)[5], and the application used to develop the ontology for this project, **Protege**, employs the OWL2 on the backend. In OWL 2, **classes** represent a set of individuals. Classes can have **subclasses**; a child class is an extension of a parent class, and individuals that fall under the domain of a child class also fall under the domain of their parent class. **Axioms** in OWL 2 can be used to relate classes to each other[5]. For example, given classes ”animal” and ”dog”, one might have the axiom stating subClassOf(dog, animal), meaning that each dog is an animal.

A **class hierarchy** represents how descriptions of ideas can go from very abstract to very specific and allow for more knowledge about the entities to be encoded in the ontology, and relationships take the form of **Object Properties**, composed of a predicate and two ideas related by this property [18]. A common example of an ontology is representing the domain of pizza[14], where one might have classes representing individual ingredients and pizzas, e.g. ”tomato sauce”, ”cheese”, ”dough”, and ”cheese pizza” and where the class ”cheese pizza” is related to each component ingredient through the relationship ”hasIngredient”. Ingredients can be assigned attributes, e.g. ”cheese” might have an attribute indicating that a portion contains 120kCal.

Supporting the ontology is a **reasoner**, which is a program that creates inferences based on the contents of an ontology and the relationships between its contents[31].

The main application will be developed in Python due to its ease of use and flexibility. In order to interact with the ontology in Python, the **Owlready2** library is used, which allows for users to interact with ontologies through Python in an object-oriented fashion [22]. **Object-Oriented Programming** is a programming paradigm with a focus on objects as the main functional units of a program, along with their associated data, methods to interact with that data, and interfaces to interact with other objects. The ontology was queried as a **Resource Description Framework** (RDF) graph, a way of representing a graph database as a form of triples where each triple is composed of an object, another object, and the relationship between them [1]. The RDF graph query language **SPARQL**[29] will be used to query the RDF Graph representing the ontology through Owlready2 in Python.

The cocktails are scraped from the trusted online cocktail archive Punch Drink through

Text Mining, the automated process of transforming unstructured textual data into meaningful structured data [23]. This process will be automated with the **Selenium Webdriver**[3], a web automation tool that allows a user to programmatically interact with web user interfaces. This information then needs to be **parsed** into cocktail objects, a process whereby which the textual data will be decomposed into a suitable and meaningful internal representation. Ingredients are matched to their corresponding representation in the cocktail ontology by using **regular expressions**, a symbol pattern matching method, to clean the data, and **Difflib**, a Python library that allows one to compute the similarity of strings, to match the text to the correct entity.
The application follows a Model-View-Controller (MVC) design pattern, where the application is composed of a model of the data (the ontology), a view where users can see and interact with the model (the front-end), and a controller, which dictates control flow dependent on the model and user interaction (the back-end) [11]. This approach is well-suited for the established object-oriented paradigm, and the graphical user interface (GUI) was implemented with **TKinter**, a Python GUI creation toolkit[4].

2.3 Current State of Knowledge

2.3.1 Cocktail Families

An ontological representation of the cocktail domain was chosen in this instance because cocktails and their relationships follow patterns that are easily captured in an ontology. Fundamentally, a ”good” cocktail maintains the balance between the five major taste sensations: bitterness, sweetness, acidity, umami, and salinity [10]. When the idea of a mixed drink was first introduced, there were only a few different combinations of these five senses that appealed to the general population enough to cement themselves as ”cocktails”, and these are preserved in history as the six cocktail families[12]. These families are:

1. The old fashioned: spirit, sugar, aromatic bitters.
2. The martini: spirit, aromatized wine.
3. The daiquiri: spirit, citrus, sugar.
4. The sidecar: spirit, fruit liquor, citrus.
5. The highball: spirit, carbonated water, citrus.

6. The flip: spirit, egg, sugar.

All of these cocktails were fundamentally successful because of the different but palatable balances of senses that they provided, and all modern cocktails are necessarily subclasses of these drinks. Thus, the structure of the cocktail evolution is suited for representation in an ontology.

2.3.2 Dietary Requirements

One challenging aspect of any entity involved in the food and drink industry is dealing with dietary requirements. Whether the requirements are made explicit to the institution through an allergen card[9] or provided to the guests' server through more ambiguous means (e.g. telling a server that one has coeliac disease, indicating a gluten intolerance), it falls upon the institution to either conform to these requirements or communicate to the customer that they are not able to satisfy those dietary requirements [15] . Thus, businesses that wish to serve customers with a wide range of dietary requirements must be able to conform to the needs of those customers in a standardized and efficient way- one way in which businesses can do this is through allergen matrices, which are ways of visualizing the inclusion of major allergens in products supplied by a vendor [15] . In the UK, there is little legislation in place protecting those with severe food allergies[16], and it is seen as a service provided by the company to be able to provide customers with the proper assurance that they are ingesting food that is safe for them to consume- the little guidance provided by the government only covers pre-packaged, ready-to-eat products- health inspections and guidelines do some to mitigate this, but allergens in restaurants still present a challenge [2] . A major component of this project is bridging the gap between vulnerable consumers and ambitious restaurateurs, who wish to publish new and exciting menus while still being able to provide an enjoyable service to excited guests. This idea is critical to the high-end, Michelin-grade restaurant industry, where the common understanding is that success comes from 51% hospitality and 49% service[25]- an interest in the customer's well-being and enjoyment falls under both hospitality and service [20].
Providing correct allergen information to the customer is only part of the service that a bar or restaurant is expected to provide- giving the customer proper recommendations based on their needs is also critical to the success of a restaurant, and large, organized knowledge bases such as ontologies provide a viable path to achieving this [27] .

2.3.3 Web Scraping

The process of getting the vast amounts of knowledge about cocktails necessary for an industrial setting could be done by hand, but this process would be time intensive and tedious. An alternative approach is automated web scraping, which can be implemented simply using the Selenium Webdriver in Python. This is an effective tool for collecting vast amounts of information stored on the web and has been deployed for research in the hospitality industry in the past[19] .

2.3.4 Ingredient Parsing

A major challenge presented in this project was being able to discern when a token scraped from the web could be reliably assigned to a class in the ontology. One approach to this issue is to utilize semantic word embeddings, an approach of determining the similarity of words based on their semantic meaning, rather than syntactic similarity, and this approach is moderately effective[6] . However, this project introduced the unique issue of brand names, which were often included in the cocktail specifications scraped from the web. Another approach is a rule-based system, which has proven effective[28], particularly in the domain of drinks; however, this approach is not public and thus could not be integrated into this project.

2.3.5 Ontology Maintenance

The medical industry is an example of an industry where ontologies are used to represent great amounts of conceptual knowledge, and the formation of these ontologies is often a joint effort between manual development and automated processes, and maintaining these ontologies can be very labor intensive and time consuming, requiring a high degree of technical expertise[8] . Many of the tools necessary to maintain ontologies efficiently are present in the Protege ontology development application[26] .

Chapter 3

Design and Implementation

3.1 System Architecture Overview

The application is centered around two structures: the cocktail ontology and the user interface. All other programs and processes are components of these or supplement the functionality of these two structures.
Figure 3.1 shows the general relationships between these components, along with their methods of communication and interaction.

3.2 Cocktail Ontology

3.2.1 Classes

The information powering the project is all stored in the ontology, which is composed of five main classes:

- Cocktail
- Ingredient
- Allergen
- Glassware
- Preparation

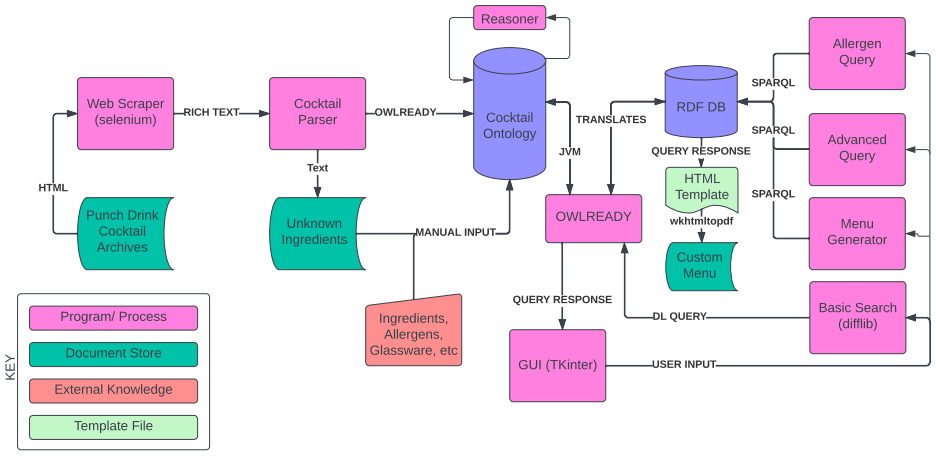

Figure 3.1: The system architecture diagram. Interaction with the ontology is driven through the GUI, which is built on the TKinter library. The query functions all act through Owlready2 in some way to access the data held in the ontology. The web scraper takes information about cocktails hosted on Punch Drink and sends this to the cocktail parser, which attempts to add cocktails to the ontology. If in the case of failure, unknown ingredients are added to a list of unknown, which along with all other ingredients, allergens, and relationships, are added to the cocktail ontology manually.

3.2.2 Object Properties

Object properties define the relationships between these classes, and these are:

- Contains
- ContainsAllergen
- IsServedIn
- IsPreparedWith

Additionally, some properties have an associated inverse property, e.g. ”IsIngredientIn” is the inverse property of ”Contains”, meaning if a some ”Cocktail” ”Contains” some ”Ingredient”, then this ”Ingredient” ”IsIngredientIn” that ”Cocktail” The domains and ranges of these properties restrict what classes are allowed to be associated with another given the specific property, e.g. the domain of ”Contains” is ”Cocktail” and the range is ”Ingredient”. The names of object properties greatly affect the ease of use when querying the ontology [33], and with this structure, queries, in particular description logic queries, are nearly formed as spoken word. If one wanted to find a list of all cocktails that have allergens, the query would be simply ”Cocktail and Contains some Allergen”.

3.2.3 Cocktail Population

There are few existing databases centered around the cocktail domain, and those resources that do exist are not reliable and do not conform to the high standards that the target audience of this application hold. One of the most trusted resources in the craft cocktail industry is Punch Drink, a blog and archive that is purposefully made for industry professionals and connoisseurs, and their extensive cocktail archive is well suited for this particular application. However, the archives are unstructured textual data, not stored in a database form, so it was necessary to design a program to mine this text data and transform it into structured data for the ontology. **Algorithm 1: Cocktail Scraping** shows the pseudocode for the scraping and population process.

Navigation

Web automation for this project was necessary because of the vast number of cocktails held in the archives and the aims of the project; a user would desire a wide range

Algorithm 1 Cocktail Scraping

```
1:  while next_page ≠ NULL do
2:      cocktail_list ← []
3:      cocktail_windows ← []
4:      for link to cocktail article on current page do
5:          Click link and append window to cocktail_windows
6:      end for
7:      for window in cocktail_windows do
8:          Open window
9:          Find cocktail name, ingredients list, and preparation by XPATH
10:         c = new Cocktail(name, ingredients, preparation)
11:         Append c to cocktail_list
12:     end for
13:     for cocktail in cocktail_list do
14:         Create new cocktail class in ontology with cocktail.name
15:         Add rdfs:comment for cocktail.ingredients, cocktail.preperation
16:         for ingredient in cocktail.ingredients do
17:             find closest matching ingredient in ontology
18:             if match is above threshold then
19:                 Add restriction cocktail Contains some ingredient
20:             else
21:                 add ingredient to list of unknown ingredients
22:                 Discard cocktail
23:             end if
24:         end for
25:         save updated ontology
26:     end for
27:     Move to next page in archive
28: end while
```

of cocktails for their menu if they wish to use this tool to innovate and spur interest. A tool commonly employed for this task is the Selenium Webdriver, which operates Chromium windows by converting operations called by the Selenium client to HTTP requests, sending these to the browser driver, and waiting for a response. Selenium was chosen for its simplicity and flexibility; the Webdriver offers a wide array of ways for searching for specific elements on a web page, which is particularly useful when specific elements are not defined simply as their own class, but specified in other ways such as by item properties (itemprop); in this instance, these specific elements can only be located by XPATH, which Selenium offers a robust solution to.
While the Punch Archives are generally unstructured, the pages are laid out in a similar fashion allowing for automation. For example, the navigation page always contains 18 drinks, and the link to each article is contained in the wrapper class "recipetease__title", so opening each window simply involves instructing Selenium to follow each link that appears in this wrapper in a new window.
Similarly, each cocktail recipe is composed of a title, an image, background information, an ingredients list, preparation, garnish, and serving instructions, and occasionally additional notes from the author, e.g. when homemade ingredients are used. The process of scraping all of this information is similar to the process of finding all of the links to cocktails- identify the matching wrapper classes for the information necessary to create a cocktail object, and copy this text. A simple Cocktail class is used to store the name, ingredients, and preparation attributes of the cocktails that have been scraped, and this simplifies the ontology population process.

Ingredient Matching

One challenge of correctly parsing the cocktail from the page into the ontology was correctly matching ingredients. Often, particularly with spirits, a specific brand of alcohol will be provided. Certain classes of liquors necessitate the inclusion of a brand name, e.g. Campari or Suze, as these are unique products that commonly appear and generally cannot be substituted. However, for other classes such as Bourbon or Gin, it is infeasible and largely unnecessary to include every brand available in the ontology, as these will be interchangeable for the most part and brand selection will be a personal preference in the vast majority of cases- notable exceptions to this are Gosling's Rum and Pusser's Rum, which appear in trademarked cocktails that must be made with these particular brands.
When presented with an ingredient such as "Appleton Estate Rum", the cocktail parser

should match the ingredient ”Rum”, not ”Applejack” or ”Apple Juice”, so it was necessary to include some logic to mitigate the influence of brand names. The solution in this instance was to search for keywords such as ”preferably”, which is often used in the recipe titles to indicate a brand preference and all units of measure when matching ingredients. For example, when matching the ingredient ”2 Oz Rum (Preferably Appleton Estate 8 yr.)”, the program transforms this ingredient to simply ”Rum” through pattern matching. Pattern matching through regular expressions is used to remove excess information from the name of the ingredient as scraped from the website, such as brand preferences or units of measure.
Another problem with ingredient parsing is alternative spellings to common ingredients- one might often see ”Rhum” instead of ”Rum”, or ”Fernet-Branca” instead of ”Fernet Branca”, so it was necessary to judge a match based on a similarity threshold instead of searching for exact matches. The Python string matching library Difflib includes this feature, so this was used to find a list of matching ingredients past a certain threshold; in this instance, a similarity score of 0.6 was required to be considered as a match. If no ingredient in the ontology matches the ingredient from the cocktail recipe, the cocktail is discarded and not added to the ontology, and the ingredient is added to a text file containing unknown ingredients so that an administrator can properly add these ingredients to the ontology.

Ingredient Population

The ingredients for the ontology were all populated by hand. One could feasibly use an existing database for the majority of non-alcoholic ingredients, but this approach would not create a proper class hierarchy. One could use an alcohol supplier/ distributor’s data sets for populating the alcoholic ingredients, but this would predominantly be branded spirits and similarly not capture the proper class hierarchy, so this approach is also not appropriate, thus manual population was necessary. Ingredients in the cocktail domain can be broadly categorized as alcoholic and non-alcoholic. Alcoholic ingredients in this ontology are either spirits, liquors, bitters, wine ingredients, or beers. An example class hierarchy for ”Rhum Agricole”, a French-Carribean Rum distilled from fresh sugarcane juice, would be ”Rhum Agricole” is a ”Rum” is a ”Sugar Spirit” is a ”Spirit” is an ”alcoholic ingredient” is an ”ingredient” is a ”thing”. A class hierarchy such as this allows for more flexibility when querying the ontology for cocktails- maybe one doesn’t necessarily just want a cocktail with a specific type of spirit in it, but a particular class of spirit, or no spirit at all.

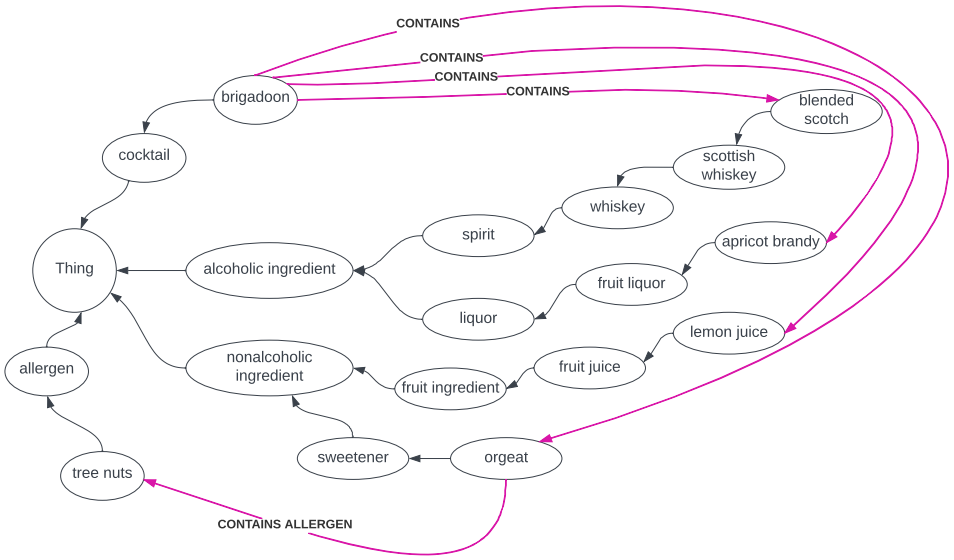

Figure 3.2: At the base of the ontology is the class ”Thing”, which everything is a subclass of. Cocktails are things, and the example element of the cocktail class provided here is a ”Brigadoom”. The ingredients of the ”Brigadoon” cocktail are blended scotch, apricot brandy, lemon juice, or orgeat, and the cocktail is related to all of these ingredients through the ”Contains” relation. Orgeat is a type of almost syrup, and that is captured through the relation ”containsAllergen”, which links orgeat to Tree Nuts.

Allergen Population

The 14 major allergens specified by the UK Food Standards Agency are celery, cereals containing gluten, crustaceans, eggs, fish, lupin, milk, mollusks, mustard, peanuts, sesame, soybeans, sulfur dioxide and sulphites, and tree nuts. These are the allergens that are reflected in the ontology, and although there do exist other food allergens, the Food Standards Agency lists these 14 as the most potent and prevalent allergens. Once these allergens were established in the ontology, a beverage supplier allergen information sheet was used to relate individual ingredients to allergens through the ”containsAllergen” relation. **Figure 3.2** shows how ingredients, cocktails, and allergens would relate given an example cocktail.

3.2.4 Ontology Queries

In order for the application to make use of the ontology, there needs to be some way for the front and back end to communicate with each other. The vast majority of the application was developed in Python, so the atypical but convenient Python OwlReady2

API was employed for the task of interacting with the ontology programmatically. OwlReady2 offers a way for one to interact with an ontology in an object-oriented style through Python. While this is a convenient solution, there are drawbacks- it is not possible through the OwlReady2 API to use a reasoner on the ontology, which removes a lot of the functionality that results from the ontology design. In order to avoid this drawback, the ontology that is used for queries for the main application is one in which all axioms inferred from a reasoner have been exported and saved as new axioms. While this does allow one to use the full ontology, this greatly hurts maintainability and robustness. If one were to scrape for new cocktails, they would be added to the new ontology very easily, but there might be aspects of these new cocktails that are overlooked by the queries until a new reasoned ontology is exported.
A common approach for querying ontologies is through the use of description logic. If one designs and names their ontology in a proper and consistent way, description logic can be a very powerful tool for querying an ontology. However, the expressiveness of description logic was not enough for certain aspects of this application, so this method was not used. Instead, the ontology is queried as an RDF graph using the Python RDFLib library through SPARQL queries. This was the best way to interact with the ontology through Python, as it allowed for complex nested queries to be written, which were necessary components for the advanced query function.

Advanced Query Formation

A notable challenge was generating a query to support the advanced search functionality. In order to make this tool as flexible as possible, it was desirable to allow users to specify what ingredients they want to include or omit to the most arbitrary degree supported by the ontology and for the users to be able to do the same with allergies. The user should be able to specify an arbitrary number of criteria for the advanced search, so it was necessary to achieve a solution that scaled well. The final solution was implemented using Python sets- the search initially finds a set of all cocktails, and then queries each specification individually, taking the intersection of the current working cocktail set if the user wants these cocktails included, and taking the difference if the user wants these omitted. This results in an O(n) time complexity, meaning the time it takes to achieve a query response is linearly related to the number of specifications, which is reasonable for this task.

Menu Generation Query Formation

When creating a balanced menu, it is necessary that there is a wide range of options available but that there is also some degree of familiarity for the customers with the ingredients. In order to achieve this, the query starts with a set of acceptable spirit bases to select from, and then randomly samples 7 of these without replacement. Then, the ontology is queried for a list of cocktails using each base spirit, and each of those lists is sampled to create the final menu. While this approach does take longer than querying for 7 random cocktails, it ensures that the types of drinks are evenly distributed and that a new menu is generated each time. One of the benefits of populating the ontology with cocktails from a trusted source is that there is a minimum standard for the drink to appear in the ontology at all, and there will unlikely be drinks included in the menu that are poorly designed or that are missing key information in terms of preparation and ingredients.

Allergen Matrix Query

When exporting the allergen matrix for a given menu, the ontology is queried for allergens contained in each cocktail. If a cocktail is found to contain an allergen, this gets noted on a Dataframe created through the Pandas library, which is a data structure similar to a two-dimensional list. Once all cocktails are searched for allergens, this Dataframe is exported as a PNG file using the savefig function included in Pandas and stored as matrix.png.

3.3 Graphical User Interface

3.3.1 Overview

The user is intended to interact with the application purely through the graphical user interface, which should act as a black box and completely hide the underlying architecture from the user. The target user groups are those interested in cocktails and bar/restaurant administrators, so the application should be able to support advanced features that a bar administrator might desire while still being able to be intuitive and simple enough to be used and appreciated by a casual user. In turn, this means splitting up the functionality of the application among multiple views, and in this case, those views are:

- A main page, where users can easily filter and find cocktails that are stored in the program, along with buttons that allow the user to navigate to the other pages.

- An advanced query page that allows for searches with an arbitrary number of specifications (where supported by the information stored in the ontology).

- A menu generation page that automatically populates a unique and balanced cocktail menu that can be exported on the spot along with compliant allergen information.

Main Page

The main page hosts the navigation buttons and provides much of the core functionality that the standard, non-expert user would want from the application. Along the right side of the main page is a list box filled with the cocktails that live in the ontology, along with a filter bar that allows the user to filter by name. This was chosen over a traditional search bar, as it is much more responsive for the user and allows the user to get an idea of what is held in the database- many of the cocktails that were scraped are not classic cocktails, but modern cocktails, and most users are unlikely to be familiar with the majority of the cocktails in the database. In order to view the ingredients and allergens contained in a cocktail, a user can click on the name, and a new page will open containing all of the details held in the ontology about that particular cocktail.
In addition to this search function, there is an allergen filtration system, which is comprised of a series of checkboxes for all of the 14 major allergens, which allows a user to filter for cocktails omitting those allergens. When this search is initiated, a new page opens up which lists all of these cocktails along with their ingredients, preparation method, and all other information held about them in the ontology.
These two features are implemented directly into the main page as they are likely to be the functions that most users will want to use- instead of creating a bloated user interface, these two features are directly available and intuitively presented, with more advanced features hidden behind navigation buttons on the bottom of the main page. This design choice conforms to Shneiderman's eighth golden rule of interface design, which states that applications should strive to reduce short-term memory load[30]. By keeping the core functionality isolated on the main page with advanced functionality isolated to their dedicated pages, users are less likely to be overwhelmed and confused by the user interface.

Certain aspects of the application, such as the advanced query page and the menu generation page, could involve noticeable loading times while the program queries the ontology due to the complexity of the query. In order to mitigate this, the loading progress is indicated to the user, so as to not confuse the user or leave them thinking that something has broken. This is consistent with Shneiderman's third golden rule of interface design, which states that informative feedback should be provided to users when an action is taken [30]. This rule is also extended to the query pages where it is possible for no cocktails to be returned- the application indicates that no cocktail matching the specified query has been found, and includes a button to return to the previous search. Aesthetically, the interface aims for consistency among all pages in accordance with Schniederman's first golden rule[30]. All pages have the same color scheme, back buttons are kept in the same location as much as possible, button colors are consistent and indicative of their functionality, and the font is consistent throughout. By designing the interface in this way, the application has a sense of continuity and is more intuitive and usable. **Figure 3.3** depicts the general layout of the main user interface.

Advanced Query Page

The implementation of the advanced query page presents a way for a user to search for drinks that conform to their exact standards. It is structured around modules that can be added to the interface by the user, which act as instructions for what ingredient or allergen the cocktail is requested to contain or omit. Because the ingredients in the ontology are organized in a hierarchical manner, this advanced query function is quite flexible. For example, one could query for all cocktails which omit sulphites, include a bitter liquor, and include a spirit that has been distilled from rice, not just cocktails that contain particular individual ingredients.
Once the query has been submitted, the user is taken to a results page, which either indicates that no cocktails matching this query have been found or lists the cocktails that satisfy the criteria.

Menu Generation Page

The menu generator is a tool primarily aimed at bar administrators. In many high-end eating and drinking establishments, it is common for the menu on offer to change daily, in response to seasonality, production levels, user taste, or a variety of other factors. Satan's Whiskers, ranked the #1 bar in the world in both 2019 and 2023, changes its

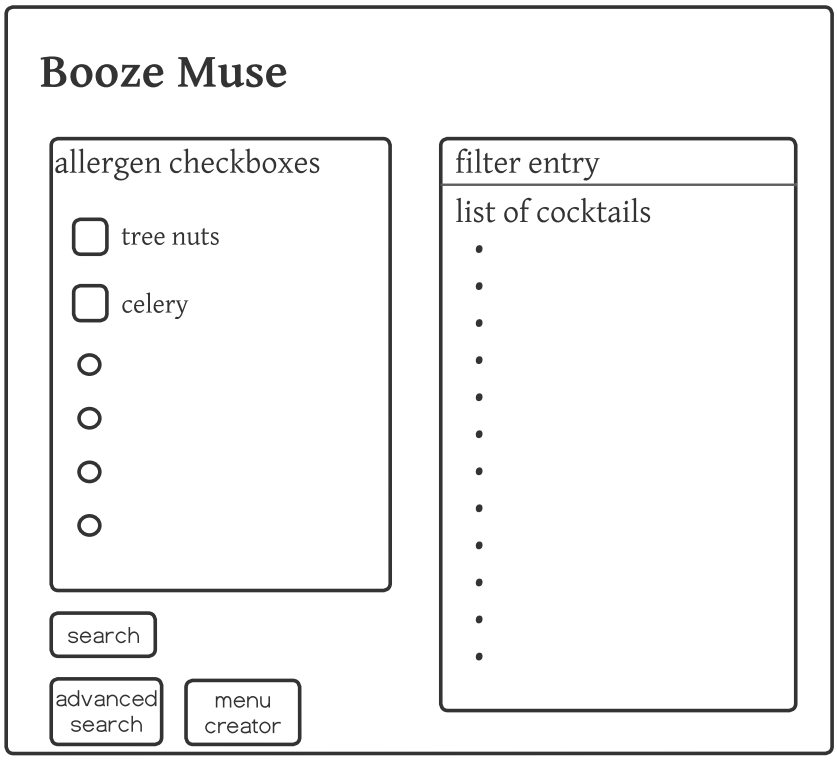

Figure 3.3: A wireframe of the main application UI. Along the left side is a section to filter by allergens, and below this is a submit button, along with additional navigation buttons. Along the right side of the main page is a list of all of the cocktails in the ontology, along with a search bar.

menu daily in this fashion, and runs a bar consultancy business that provides advice on how to design a balanced and successful cocktail menu; their guidance has been integrated into the menu generation algorithm. If a bar administrator wanted their bar to reach these high standards but did not have the time or resources to create a new menu from scratch every day, they could rely on this menu generation tool to quickly create a workable menu for them, complete with allergen information and drink specifications. After the menu generator selects its drinks, these are displayed on the user interface, and included in the UI are buttons that allow for one to export this menu along with its allergen information.
When the menu is exported as a PDF the cocktails populate a stylized HTML menu template, which allows for the menu to be printed out and deployed into service nearly instantly. This is done by using the Jinja2 Templating Engine, which allows one to create an HTML document with specific characters to indicate where Python variables should be substituted in. After the menu variables such as cocktail name and ingredients are substituted into the appropriate place in the menu template, the pdfkit library and wkhtmltopdf tool are used to export this HTML file as a PDF file. wkhtmltopdf is a command line tool that converts HTML documents to PDF documents, and the pdfkit library allows one to publish this PDF to the filesystem.

3.3.2 TKinter

The user interface is designed using TKinter, a Python package created for designing user interfaces. The design follows an object-oriented programming (OOP) paradigm, in which the main application is a class, and all objects contained within the user interface are children of this class[24]. Data is encapsulated into classes, and classes have interfaces for interacting with others. Additionally, child classes inherit certain attributes and functionality from their parents[24]. This is similar to a model-view-controller (MVC) design pattern, with the controller equivalent to the main application class and the views being the children of this main application class[11]. The model in this instance is the ontology, which the controller interacts with through OwlReady2. The OOP design approach to the user interface was chosen because many of the different pages share resources, and it would not be as efficient to make each individual page act independently for shared queries. For example, each page needs a list of the available cocktails and a list of allergens. Most pages also require an ingredients list. By using an OOP approach, the children of the main window all inherit necessary attributes and reduce application latency. The ”model” aspect of this application

is data obtained through the OwlReady2 interface- the main program body, the controller, forms queries based on user requests, queries the model through OwlReady2 and forwards these responses to the view for the user to interact with.

Chapter 4

Testing Methodology

4.1 Usability Testing

The target audience for this application is those interested in cocktails and bar administrators. These demographics are not expected to have much technical expertise or experience, so one evaluation method for the application will be usability. The user interface is intended to allow both simple and advanced interactions with the knowledge held in the ontology while completely hiding the underlying architecture from the user, and a usability test would be one way to see how completely this has been achieved. The evaluation will be structured around tasks provided to the users and a survey with questions for the users with feedback sections.

4.1.1 Usability Trial and Survey

Tasks provided to the user will ask them to complete actions that compose the core functionality of the application. Examples include creating an advanced query that conforms to certain specifications and naming a cocktail that is provided as a result of this, or creating a new cocktail menu, exporting the menu as a PDF document, then finding and viewing this menu. These are all tasks that should require very little computer experience, so the feedback should reflect this and focus on the actual usability and intuitiveness of the program.
The tasks and questions will focus on

1. Cocktail Search Functionality

2. Allergen Filtering Functionality

3. Menu Generation
4. Menu and Allergen Matrix Exporting
5. Advanced Query Formulation
6. Overall Usability and Aesthetics

These particular tasks are chosen to represent the overall usability of the application. The processes allow users to experience and test all functional aspects of the GUI, providing useful insight into the overall user experience. Any aspects of the application that are confusing, too technical, or broken should be exposed though the completion of these tasks.
The survey following the tasks will include questions to assess if the task has been completed with the expected results, assess how simple and intuitive this process is, and enquire of the user ways the functionality in question could be improved.
A component of the usability testing will be dedicated to feedback about the application. Some feedback questions are designed to gain insight into the general usability of the application- this should convey what aspects of the program are confusing, discover any unexpected or erratic behavior, and discern what can be improved upon in future iterations. Another component will focus on the aesthetics of the application- the impact of aesthetics stretches far beyond making some products look beautiful, and it is important for the application to have functionality in mind [32]. Additionally, if this tool were to be deployed and used in the real world, it would be important for the menu component to be presentable and interesting- much of the functionality would be lost if the application exported a menu that was unusable for some reason.
The thinking aloud method will be employed in this usability test, which asks those testing the software to literally 'think aloud'- this method allows for the evaluator to see the thought process of the user, and determine what aspects need to be improved even if the user does not explicitly state this in the feedback form. The thinking aloud method can aid in requirements analysis[21], and one of the aims of this evaluation is to determine what further features can be added to the application to improve it.

4.1.2 Ethical Considerations for Usability Study

The usability study will be completely anonymized, survey participants will participate on a basis of informed consent, and participant groups are limited to peers and colleagues. Survey participants will be requested to have some level of knowledge

about cocktails, but no technical skills will be required to complete the survey, and thus will not cause significant levels of distress. No personal information will be requested from the survey participants.

4.2 Technical Testing

This application has various technical aspects that could introduce issues that may not present themselves in the usability tests. For example, users will not be interacting with the ontology, so they will not have any way of providing feedback on the robustness, accuracy, and scalability of the application. To address this, various other tests will be introduced to evaluate these aspects.

4.2.1 Robustness

The robustness will be evaluated will be through the trial addition of new cocktails and ingredients. The process of updating the ontology with new ingredients and cocktails should be as simple as possible if a robust system is to be developed. Trialing this process will provide insight into the number and complexity of steps that are necessary to introduce new entities into the ontology, which is something an administrator would likely desire to do.

4.2.2 Accuracy

The tastes of people that dictate new cocktail creations change with time, and it is certain that new cocktails will be introduced that include ingredients that will not be included in the existing ontology. While the cocktail parser does create a list of unknown ingredients, it is necessary to evaluate how well ingredients are attributed to the correct class, and how well unknown ingredients are spotted at dealt with accordingly. It is important for the parser to accurately categorize ingredients; if it fails to do so, incorrect allergen information could be provided which could have a devastating impact on customers and administrators. A manual overview of the ingredient parsing process will be employed to assess how accurately ingredients are attributed. In addition to this, potentially catastrophic misclassifications will be noted and investigated- examples of these could include an alcoholic ingredient being classified as non-alcoholic or an ingredient containing an allergen being classified as not containing that allergen.

4.2.3 Scalability

It is critical for this tool to be simple and intuitive, and a snappy user interface is integral to this. In order for the UI to achieve this, the algorithms implemented for searching through the cocktails need to scale well. This will be evaluated by recording loading times with varying numbers of cocktails and ingredients in the ontology and should indicate how well the application scales with the number of cocktails in the ontology. In order to time this process, the Python time module will be embedded in the code for the controller, and the number of cocktails that are loaded will be changed to compare times.

Chapter 5

Results

5.1 Final Application

The final application incorporates the main search functionality, allergen search, menu generation with menu and allergen matrix exporting, and advanced query formulation into one user interface. The cocktail search and allergen search features are present on the main page, with buttons on the bottom of the page navigating to the more advanced features. **Figure 5.1** shows the user interface in its final form.

The application uses a consistent layout, color, and font size throughout to make the user experience as seamless as possible.

The menu generation tool is able to output highly stylized and digestible menus through a simple interface, and also output the accompanying allergy matrix. **Figure 5.2** shows an example menu generated by the program, and **Figure 5.3** shows an example allergen matrix.

The cocktail ontology in its final form is composed of 134 cocktails, 14 major allergens, and 214 ingredients. All ingredients and their relations to their respective allergens were populated manually through the Protege interface, and all cocktails were populated automatically through Owlready2.

5.2 Usability Results

5.2.1 Survey Participants

Prior to participating in the survey, all users were walked through an informed consent process.

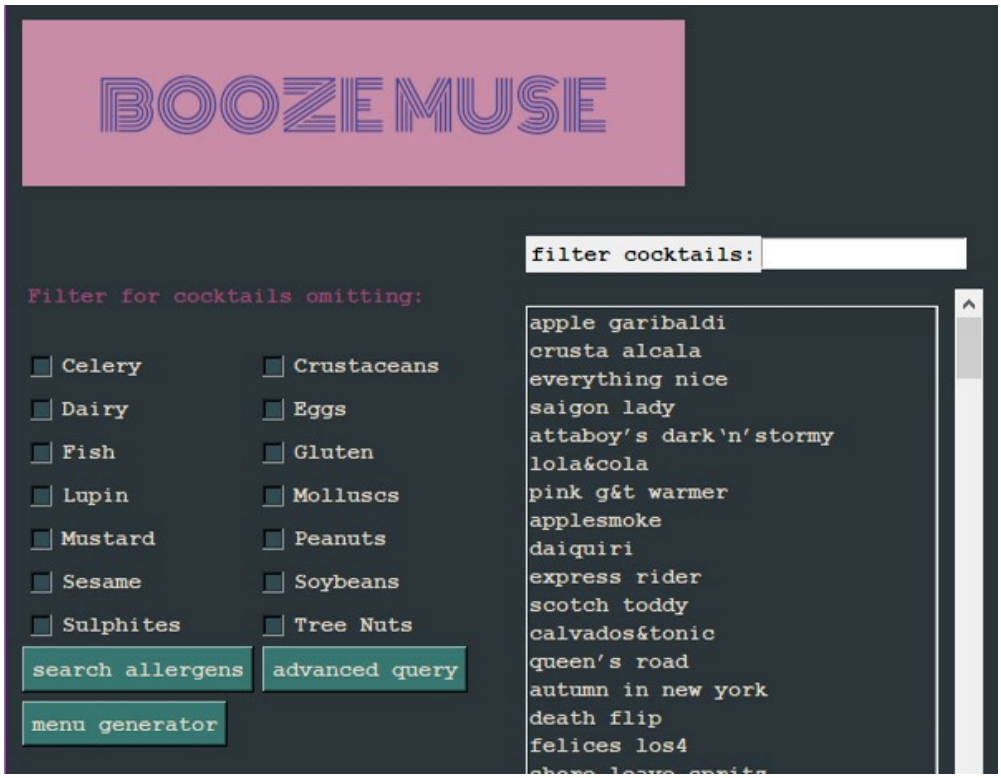

Figure 5.1: The booze Muse main interface. Along the right side is a list of cocktails contained in the ontology, accompanied by a search bar to filter through them. Along the left is a basic checkbox system for filtering for cocktails omitting specific allergens. Along with the ”search allergens” button, there are buttons to navigate to the menu generation page and advanced query page.

BOOZE MUZE MENU

golden years

1 1/2 ounces rice-aroma baijiu, preferably Vinn 1/2 ounce Scotch whisky, preferably Compass Box Artist Blend Scotch 1/4 ounce maple syrup 6 drops Bittermens mole bitters 3 drops Red Boat fish sauce 1 dash Bitter Queens Shanghai Shirley five-spice bItters

Add all ingredients to a mixing glass and stir briefly with ice. Strain over a large cube in a rocks glass.

more supreme

1 1/2 ounces rhum agricole, preferably Rhum Clément Select Barrel 3/4 ounce lime juice 1/2 ounce cane syrup 1/4 ounce Campari

Combine all ingredients, except Campari, in a mixing tin and shake with ice. Strain into a chilled cocktail glass. Slowly pour Campari against the inside edge of the glass. Garnish with freshly cracked black pepper.

Garnish: freshly cracked black pepper

dirty old man

3/4 ounce tequila, preferably Milagro 3/4 ounce mezcal, preferably Ilegal 3/4 ounce red chile-infused Aperol (see Editor's Note) 3/4 ounce lime juice 3/4 ounce pineapple juice 1/2 ounce agave nectar 2 dashes Peychaud's bitters

Combine all ingredients in a cocktail shaker and shake with ice. Strain into a rocks glass. Garnish with a dehydrated lime wheel.

Garnish: dehydrated lime wheel

ned king's gem

1 ounce Cognac, preferably Pierre Ferrand 1840 1 ounce Jamaica rum, preferably Appleton Signature 3/4 ounce lime juice 1/2 ounce pineapple gomme syrup or simple pineapple syrup (see Editor's Note) 1 dash Angostura bitters

Combine all ingredients in a shaker. Add ice and shake until chilled. Strain into a Nick & Nora or other stemmed cocktail glass. Grate cinnamon over the top of the drink.

Garnish: freshly grated cinnamon

red queen

3 ounces red wine 1 ounce bourbon 3/4 ounce lemon juice 1/2 ounce simple syrup 1/2 ounce Amaro Braulio

Combine the wine, bourbon, lemon juice, simple syrup, and amaro in a cocktail shaker. Add ice, shake, and strain over crushed ice into a tall collins glass. Garnish with lemon wheels, fresh thyme, sage, and rosemary.

Garnish: lemon wheels, fresh thyme, sage, rosemary

earthquake

2 ounces Cognac, preferably Hardy VSOP 1 ounce absinthe, preferably Lucid

Combine all ingredients in a mixing glass over cracked ice and stir until chilled. Strain into a chilled Georgian punch glass. Garnish with expressed lemon peel.

Garnish: lemon twist

allegory's pisco sour

2 ounces Caravedo pisco 1/2 ounce lime juice 1/2 ounce lemon juice 1/2 ounce gomme simple syrup, (1:1, sugar to water) 1/2 ounce rich simple syrup , (2:1, sugar to water) 1 egg white 1/2 ounce soda water, preferably Topo Chico

Combine all ingredients, except for the soda, in a tin with ice and shake for 7 seconds. Strain the ice, add the soda water to the tin, and reverse dry-shake the cocktail for 10 to 15 seconds. Fine-strain into a coupe glass. Dash Angostura bitters over the top of the cocktail.

Garnish: Angostura bitters

This menu was automatically generated by the Booze Muse bar administration tool. All cocktail recipes are courtesy of punch drink

Figure 5.2: This is a menu of 7 cocktails automatically generated by the application and exported as a PDF.

	Celery	Crustaceans	Eggs	Fish	Gluten	Lupin	Dairy	Molluscs	Mustard	Peanuts	Sesame	Soybeans	Sulphites	TreeNuts
crusta alcala	n	n	n	n	n	n	n	n	n	n	n	n	n	n
glasgow razor blade	n	n	n	n	n	n	n	n	n	n	n	n	n	n
dirty old man	n	n	n	n	n	n	n	n	n	n	n	n	n	n
nightstand	n	n	n	n	n	n	n	n	n	n	n	n	n	n
banana spider	n	n	n	n	n	n	n	n	n	n	n	n	n	n
jelani johnson's vesper	n	n	n	n	n	n	n	n	n	n	n	n	Y	n
mai tai	n	n	n	n	n	n	n	n	n	n	n	n	n	Y

Figure 5.3: An example allergen matrix, as output by the application. Along the left column are the names of cocktails on the menu, and along the top row are each of the 14 allergens. The cell in which a cocktail and allergen intersect indicates whether or not that cocktail contains that allergen.

Users were told what the purpose of the application was, and what is hoped to be gained from their participation. Additionally, the estimated time of the survey and the type of tasks that would be asked of them was laid out. After this, users were informed that the survey was completely voluntary and that they could exit from the survey at any point.

Users were told that their results will be completely anonymous, but that audio will be recorded for the duration of the survey. The concept of ”think aloud” usability studies was explained to them, and they were told that after any additional notes were taken from the audio recordings, the recordings would be deleted. Additionally, the recordings will not be shared with anyone, and not be subject to long-term storage or cloud processing.

In total, there were 6 survey participants. Two of these fall under the user category of ”bar administrator”, and are people who have experience managing bars, developing menus, and making drinks. The other four people are peers who have a moderate interest in cocktails; none of these people have any professional experience in a bar or restaurant setting. This split of participants should provide a balance of technical insight on the cocktail aspects of the project, which is what the bar administrators are likely to focus on, and the functionality and usability, which the people without bar experience are likely to focus on.

5.2.2 Task-Based Questions and Feedback

Cocktail Search

Users were easily able to search through the list of cocktails to find a specified cocktail. They could find the ingredients and allergens contained in this cocktail consistently. Some provided feedback indicating the coloring and positioning of the search bar were

not intuitive.

Allergen Search

All users found it ”Very easy” to search for cocktails omitting specific allergens, and all were able to complete this task correctly.
One user commented on the coloring of the title for the allergen search functionality, noting that the color made it difficult to read, and also said that the loading time was too long. Another user commented on the format of the search results, claiming that the window was too small, making readability difficult, and that query results for this window size might better be presented as slides instead of a scrollable window. Another user said that not all of the buttons were the same size, as they were fit to the text, and this hurt the aesthetic value.

Menu Generation

All users found the process of generating a new menu ”very easy”. However, they did comment on ways that the menu generation function could be improved. One user wanted more descriptions to be provided with the cocktails, such as food pairings and descriptions of what type of cocktail was provided, such as ”aperitif”, ”digestif”, ”fruity”, etc.

Menu and Allergen Matrix Exporting

All users found the menu exporting function simple to use. However, all desired more customizability from the menu generator. The users indicated that something such as being able to change the color or font would be desirable. Additionally, users said that the current menu appeared cluttered, and in some cases the drinks put on the menu stretched beyond the template, providing them with an ugly menu. Additionally, users with experience writing menus commented on the inclusion of the measures of ingredients on the menu, saying that it would be preferential to just include the ingredients. **Figure 5.4** shows a menu generated by one user that was particularly cluttered and malformed.
Users had issues with the allergen matrix exporting, primarily noting that there was no choice of where to export the matrix to and what to name it as, which was confusing. Additionally, the exporting of the allergen matrix took a long time and provided no update to the user as to when it was loading and when it was complete, leaving all users

BOOZE MUZE MENU

golden years

1 1/2 ounces rice-aroma baijiu, preferably Vinn 1/2 ounce Scotch whisky, preferably Compass Box Artist Blend Scotch 1/4 ounce maple syrup 6 drops Bittermens mole bitters 3 drops Red Boat fish sauce 1 dash Bitter Queens Shanghai Shirley five-spice bItters

Add all ingredients to a mixing glass and stir briefly with ice. Strain over a large cube in a rocks glass.

the spaghetti incident in the jungle

1 1/2 ounces rum, preferably El Dorado 8 rum 1 1/2 ounces pineapple juice 3/4 ounce Aperol 1/2 ounce lime juice 1/2 ounce stout syrup (see Editor's Note)

Combine all ingredients in a cocktail shaker with ice, and shake until chilled. Strain into a double Old-Fashioned glass. Garnish with a lime wheel with two pineapple fronds sticking through to rest on the rim of the glass.

Garnish: lime wheel, pineapple fronds

jelani johnson's vesper

2 ounces vodka, preferably Absolut 1/2 ounce gin, preferably Tanqueray 10 1/2 ounce Cocchi Americano 1 dash orange bitters (optional)

Combine all ingredients in a mixing glass with ice and stir until chilled. Strain into a Martini or coupe glass. Garnish with lemon twist.

Garnish: lemon twist

attaboy's dark'n'stormy

2 ounces black rum, preferably Goslings Black Seal 3/4 ounce sweetened ginger juice (see Editor's Note) 1/2 ounce lime juice 3 ounces soda water

Garnish: candied ginger

Whip shake first three ingredients with a small handful of crushed ice, until you can no longer hear the ice in the tin. Open tin and pour in the club soda. Pour the drink into a chilled Collins glass over an ice spear. Garnish with candied ginger. Serve with a metal straw.

tequila matador

1 ounce tequila 2 ounces pineapple juice 3/4 lime juice

Combine all ingredients in a mixing tin and shake with shaved ice. Strain into a chilled cocktail glass.

deke dunne's widow's kiss

1 ounce apple brandy, preferably Republic Restoratives Chapman's Apple Brandy 3/4 ounce VSOP Cognac, preferably Rémy Martin scant 1/4 ounce yellow Chartreuse scant 1/4 ounce Bénédictine 2 dashes Angostura bitters

Combine all ingredients in a mixing glass. Add ice and stir to chill. Strain into a chilled Nick & Nora glass.

calvados&tonic

2 ounces ounces Roger Groult 3 Year Calvados Pays d'Auge 4 ounce Fever-Tree Aromatic Tonic Water

Build the ingredients except the garnish in a chilled Collins glass or Burgundy wine glass over ice. Lightly stir, and garnish with a lemon twist.

Garnish: lemon twist

This menu was automatically generated by the Booze Muse bar administration tool. All cocktail recipes are courtesy of punch drink

Figure 5.4: This menu includes text spilling over the bottom of the template, an error that occurs when the descriptions of the cocktails are too long.

Celery	Crustaceans	Eggs	Fish	Gluten	Lupin	Dairy	Molluscs	Mustard	Peanuts	Sesame	Soybeans	Sulpl
n	n	n	n	n	n	n	n	n	n	n	n	
n	n	n	n	n	n	n	n	n	n	n	n	
n	n	n	n	n	n	n	n	n	n	n	n	
n	n	n	n	n	n	n	n	n	n	n	n	
n	n	n	n	n	n	n	n	n	n	n	n	
n	n	n	n	n	n	n	n	n	n	n	n	
n	n	n	n	n	n	n	n	n	n	n	n	

Figure 5.5: In some instances, the left and right sides of the allergen matrix were cut off. This was observable when the names of the cocktails in the menu were particularly long.

confused as to if they had made an error. In cases where the names of the cocktails were long, this text was cut off by the allergen matrix, detracting from readability. One user suggested that instead of using ”Y/n” to indicate if a particular drink had a particular allergen, readability would be improved by simply using a checkbox where an allergen was present. **Figure 5.5** shows an example of the unreadable, bloated allergen matrix that some users exported.

Advanced Query

Users, when asked how simple the process of using the advanced query was, were split evenly between ”Moderate”, ”simple” and ”very simple”. All users were able to complete the task but commented on the length of the ingredients list they were provided with. One user suggested introducing a search bar into the filter in order to make the process easier, or an autocomplete feature. Additionally, users commented on not being able to remove a filter if they added an extra filter field by accident- they had to return to the main interface and re-enter the advanced query interface if they wanted to get rid of this accidentally added filter.

5.2.3 General Feedback

Usefulness and Practical Applications

Users indicated that the Menu Generator function in particular was exceptionally useful and practical, and claimed that the simplicity of this feature was a strong point. They also commented on the usefulness of not having to memorize large lists of cocktails along with their allergens and stated that this would be an extremely useful tool

to have when working on a bar. Users without experience working on the bar stated that they would have a better customer experience knowing that their bartender would be able to craft a drink to their specifications without as much pressure if they had this tool.

Areas of Improvement

The two areas of improvement that users expressed were in the general UI and lack of customizability. The UI aspects that users wanted to be improved were feedback for actions, placement of the search bar, and overall readability. One user with bar administration experience noted that they would like to be able to upload their own menu template to be used for menu generation, and another expressed a desire to be able to change the font, title, and colors of the menu.

Overall Intuitiveness

66% of users found the user interface ”intuitive”, while 33% of the users found the user interface ”very intuitive”. The deciding factor in this was expressed as the lengthy loading times.

5.3 Technical Evaluation

5.3.1 Parsing Accuracy

The overall measured classification was 90.2% for matches that were technically correct given the information scraped from the website, and 76.7% for exact matches. Here, the term ”technically correct” indicates that the ingredient was classified as an ingredient label that was not necessarily wrong, but there existed in the ontology a more closely matching label that would be more correct.

5.3.2 Maintainability Trial

The maintainability trial included introducing a new cocktail to the ontology and having this fully integrated correctly into the GUI. The process took 1 minute and 30 seconds and involved 7 major steps, all of which were highly technical and could not be expected from an average user. These steps involved

1. Opening the cocktail ontology in Protege

2. Creating a new cocktail
3. Writing class restrictions for ingredients
4. Saving the ontology
5. Exporting the fully reasoned Ontology
6. Opening the main application code
7. Updating the code to reference the new ontology

All of these steps require the maintainer to have a high degree of technical proficiency and familiarity with this application and fail to abstract the user away from the ontology data structure. Using the cocktail scraper, the addition of new cocktails is much simpler but still requires the user to open Protege, export the reasoned ontology, and redirect the code to the new ontology.

5.3.3 Scalability Tests

The scalability of the application was evaluated first by measuring the query completion time for the allergen search functionality with varying numbers of cocktails in the ontology. This was tested for 0, 1, 2, and 3 filters applied to the search, for numbers of cocktails in the range of (10,130), measured at increments of 10. **Figure 4.6** shows the results of this evaluation.

Following this, the menu generation feature was tested for scalability for 40-130 cocktails, in increments of 10. The same range of cocktails was not used for this evaluation because the menu generator could not create a menu with less than 40 cocktails. **Figure 4.7** shows the results of the menu scalability testing.

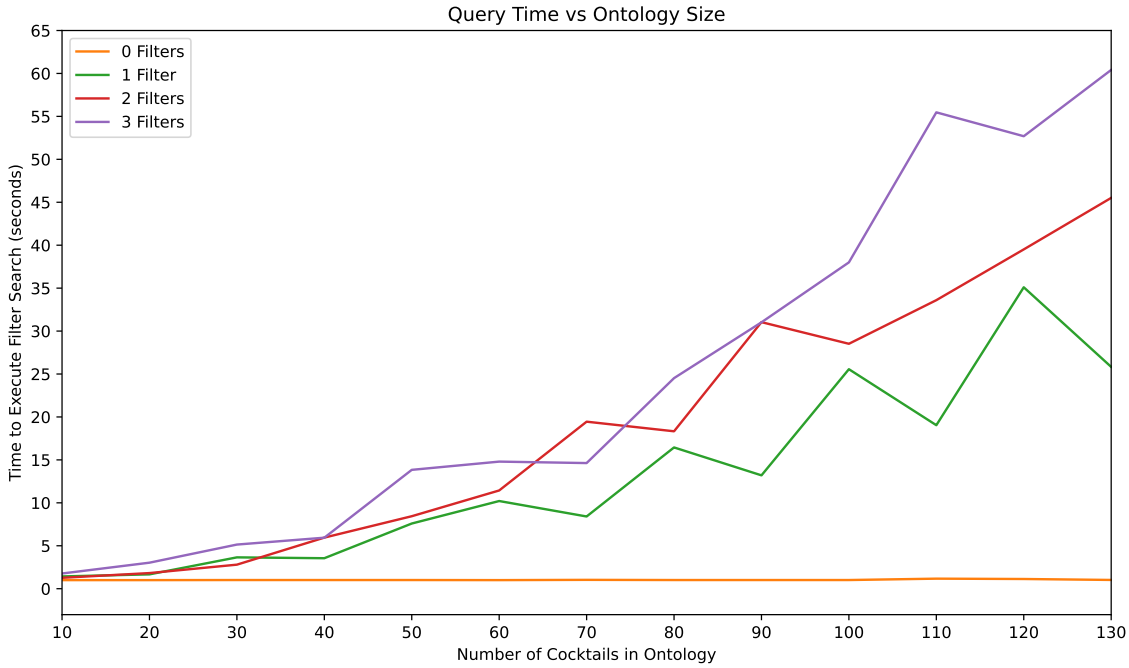

Figure 5.6: Results of scalability testing for allergen filtering query results. Generally, as the number of cocktails held in the ontology increases, so does the time to execute the query. Generally, the number of filters applied to a query is positively correlated with the time to execute the query.

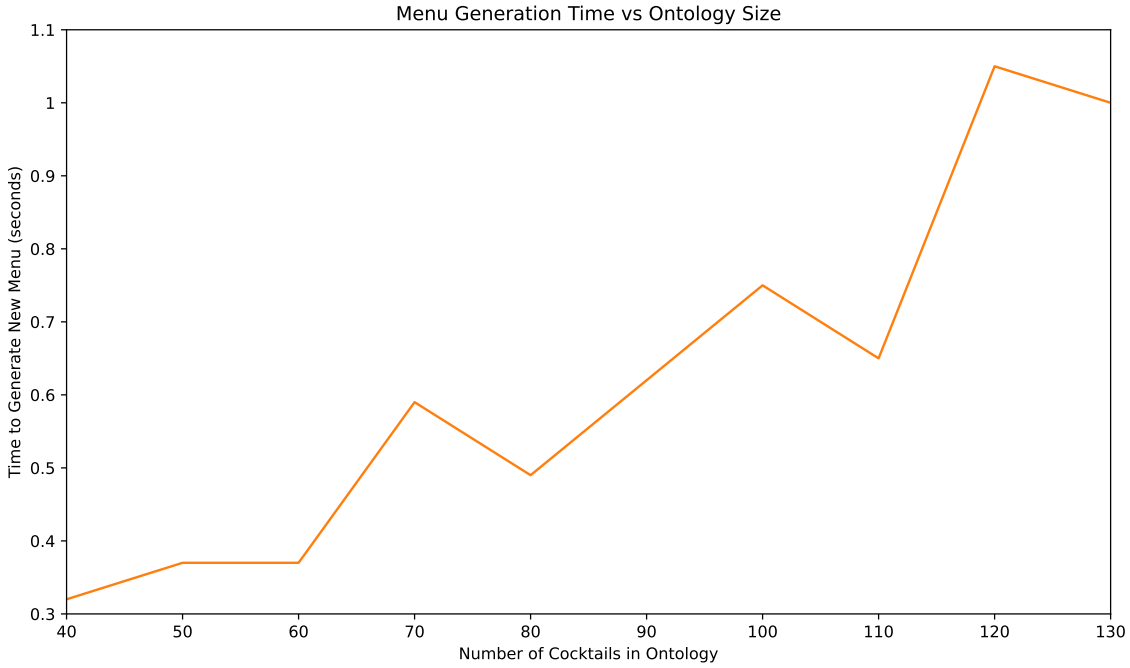

Figure 5.7: As the size of the ontology increases, so does the loading time for the menu generation. This process, however, takes a much more reasonable amount of time, with the longest loading times being just over 1 second.

Chapter 6

Discussion

6.1 Discussion of Usability Study

Cocktail Search

Users found it extremely simple to search through the cocktails in the list, but the placement and style of the search bar were confusing for some. This makes sense, as the search bar was white in color and placed slightly above the list of cocktails it was filtering through. If the search bar had a color more similar to the cocktail list and was placed directly on top of this bar, it would provide a much more intuitive user experience.

Allergen Search

All users were able to complete this task, but many were confused and off-put by the loading times. This allergen search is one of the core functionalities of the program, so a short loading time is desirable for this task. The GUI gave feedback to the user that the query results were loading, but a lack of a progress bar paired with an unresponsive GUI during these loading periods was confusing. Integrating a progress bar would alleviate the long loading times somewhat, by allowing a user to know that some progress is being made- otherwise, they might simply think that the program has crashed because it is unresponsive during this time. Additionally, the text denoting the ”allergen search” section of the page was not very readable, as indicated by the users. The text is purple with a dark blue background, and the font size is 12, so increasing the font size and choosing a higher-contrast color for this text would improve the readability drastically.

Menu Generation

Overall, users found the menu generation feature to be the most impressive feature included in the application. Users commented on its usefulness to them as bar administrators or indicated that they could see this feature being useful for a bar administrator. The loading times for the menu generation were low, and this included a progress bar, which many users commented on the helpfulness of. Some users indicated that a future improvement would be a system to recommend food parings for the cocktails, which is feasible but outside the scope of this project. More relevant suggestions included the inclusion of words and key phrases to denote the type of drink that was being recommended, which is feasible. For example, if a drink has some fruit juice and sweetener in it, it could be classified as a ”fruity” cocktail, or if a bitter component is involved it could be classified as an ”aperitif”. These additional improvements would make the application much more usable and helpful for those using the application who are not bar administrators, but those looking for cocktails to make with the ingredients that they happen to have at home. The average person may not know what specifically makes a cocktail better before or after dinner, and this application could indicate that if designed in this way.

Menu and Allergen Matrix Exporting

The use of the Tkinter filedialog object, which presents the user with the operating system’s standard interface for saving files, made the process of exporting the menu as a PDF simple and familiar to the users, and no one struggled with this aspect. However, this same process was not present for exporting the allergy matrix, and this confused users who were expecting a similar dialog window to open up. Instead, the matrix was saved to a default location with a long loading time, during which the GUI was frozen, and no indication that the exporting process had been completed. Users should not feel as if they have broken something while using the application when they have not, and they felt this way due to the lack of consistency in the GUI and a lack of informative feedback. Providing the user with the same dialog window as that for the PDF and an indication that the allergen matrix was loading would remove these issues and improve the overall user experience [17].

Some users stated that they wanted more customization with the menu. While they could do this by modifying the HTML template or creating their own, this is not reasonable to expect from the average user. A solution to this would be to create variables

in the HTML template dictating the color, font size, etc, and integrate these as dropdown menus in the menu generation page, providing users with a higher degree of customization.
Multiple users had a bug with the presentation of the allergen matrix, with certain key portions of the image being cropped out. If the user is not able to read the cocktail that the row in the allergen matrix is corresponding to, the matrix is functionally useless, unless no cocktails contain any allergens. This bug was due to the length the names of some cocktails being too long. Additionally, the allergens were indicated with a ”Y/n” system, which made readability difficult for a matrix of 98 cells. A more appropriate approach would be a coloring system, where cocktails containing a specific allergen are highlighted as red, with everything else being white or green, or a system in which a tick indicates a cocktail containing a specific allergen. It is important for this allergy matrix to be easily readable by guests and service workers, and overwhelming these people with too much information could lead to mistakes that have catastrophic consequences.

Advanced Search

All users were able to easily use the advanced search functionality, add search filters, and select these appropriately. However, the list of ingredients that one was presented with was universally stated to be overwhelming. Instead of providing a user with a dropdown list of hundreds of ingredients to search through, introducing some search functionality would be a much more elegant and efficient approach. One of the benefits of providing a user with a dropdown list is that users are presented with all of the possible options to filter cocktails by- many users expected that only specific ingredients could be queried for, and were surprised that more general filters such as ”alcoholic ingredient” were valid search filters. While this does improve functionality, it does not improve usability.

6.2 Discussion of Technical Results

Parsing Accuracy

The most common errors that caused a ”technically correct” classification were among the ”bitters” and ”vermouth” categories. ”Bitters” is a class of alcoholic ingredients that has many different types with very distinct characteristics, and these range from

”Orange Bitters” to ”Tobacco Bitters”. Except in the case of ”Angostura Bitters”, all bitters were classified under the ”bitters” class. In 1 of the 34 total misclassifications of bitters, the ingredient in question was ”walnut bitters”, and this would not have appeared as an allergen for this cocktail.

The vermouth misclassifications occurred as most cases indicated whether it was ”sweet”, ”dry”, or ”Bianco” vermouth, and only ”Bianco Vermouth” was correctly classified. While the allergy information is still correct, as all of these include sulfites, including the parent class ”Vermouth”, these are very different products and may not give the user the result that they expect.

Among those misclassified, the most common error was classifying ”Pineapple Juice” as ”Apple Juice”. While this does not lead to any catastrophic errors, it is an incorrect classification. The reason for these incorrect classifications has to do with the usage of the Difflib package. When the word similarity is found, the ingredient from the recipe page is passed to the word similarity function, along with a list of all of the possible ingredients sourced from the ontology. Because the ”apple juice” ingredient is fully contained in the string ”pineapple juice”, both the ”pineapple juice” and ”apple juice” ingredients were returned as being correct matches. Instead of searching through the potential matches and finding the best one, the algorithm simply finds the word with the closest match and takes this. Since both ”pineapple juice” and ”apple juice” have the same similarity score, apple juice is the first item in the list of similar words and gets chosen. This issue is also present in the classifications of vermouth.

Bitters are often only partially categorized right because the ontology does not contain many of the bitters that were used- only the common ones were included. This is an issue because the ”bitters” class still appears in the ontology, and while ”walnut bitters” truly are ”bitters”, the lack of specification detracts from the robustness of the application. As a result of misclassifications like these, meaningful allergen information is missed, which could have catastrophic effects. While it is unlikely that a bartender using this application would knowingly put walnut bitters into a drink that is supposed to be free from tree nuts, this information is not conveyed in the allergen matrix that a customer might be presented with and could mislead them into ordering something that is unsafe.

Because of this method of classifying word similarity, the algorithm is sure that ”walnut bitters” corresponds to an ingredient that is contained in the ontology because of the overlap of the word ”bitters”. While this aspect of being able to classify and search by very vague or very specific terms was allowed to persist in order to allow for more

broad queries, e.g. if one simply wanted a drink that contains any type of bitters, this function is not desirable during the parsing process, where unknown ingredients and their associated cocktails should be discarded. Additionally, a more expansive list of ingredients would have improved this process, as among those cocktails viewed by the scraper, only around 20% were able to be added to the cocktail ontology.
A better approach to word matching would have been through a semantic similarity model. This could largely detract from the speed at which the scraping could be achieved, but would likely get better results. ”Pineapple juice” would not be misclassified as ”apple juice” using this model, as the similarity of words would not purely be related to their lexical similarity. In addition, less preprocessing would have to be done in order to get rid of brand names. For example ”Appleton Estate Signature Rum” would get classified as ”apple juice” rather than ”rum” using the Difflib package, had the brand labels not been stripped from the ingredient, but a semantic similarity pipeline, such as that offered by HuggingFace that incorporate the Bert large language model, would likely have correctly classified this as ”rum” with no preprocessing.
Overall, the parser performed well, with reasonable classification accuracy. Had more ingredients been included in the ontology, particularly for bitters, the accuracy score would be very high, but because this was not the case, some critical errors were made by the parser. While employing a pre-trained large language model and utilizing semantic word embeddings would have likely provided better results, the additional complexity, computational overhead, and time to employ this tool would not have been worthwhile.

Scalability Testing

This program offered poor scalability results, achieving long loading times with relatively few cocktails in the ontology and simple queries, particularly during the allergen search task. This is likely due to the overhead introduced by the Owlready2 API and the resultant query formation. The choice of developing this application primarily in Python left Owlready2 as the only way of interacting with the ontology, and certain aspects of this API introduced major challenges. Owlready2 is not able to use a reasoner and perform queries on the ontology. Additionally, Owlready2 interfaces with the Java Virtual Machine (JVM) rather than interact with the ontology directly, as the OWL API for Java would. These two factors mean that the only way to gain the necessary functionality using Python is by exporting the fully reasoned ontology, which detracts

from the maintainability of the application and largely bloats the ontology with unnecessary information. In order to access the ontology and get data necessary for the application through Owlready2, the queries needed to be SPARQL queries, which utilize the RDFLib library. The fully reasoned ontology is converted into an RDF graph database, and then the RDFLib library is utilized in order to perform these queries; RDFLib has notably worse performance than its Java and C# counterparts for interacting with RDF graph databases [7]. The SPARQL queries for the allergen search are highly nested, and this nested structure vastly increases the time complexity of the application. If description logic queries were able to be employed for this application, the search process would have been much more efficient.

Maintainability Trial

The complexity of updating the ontology with new ingredients demonstrates the lack of maintainability for this application. If there was a system administrator in charge of updating and managing the ontology, this would be feasible, but it is desirable for a user to be able to update and manage the knowledge stored in the program as they see fit. Because the program only works once the fully reasoned ontology has been integrated into the code, there is no way for updates to the ontology to completely work unless this happens, and doing this programmatically with Python is infeasible because of the inflexibility of the Owlready2 API. It would be possible to allow users to create new cocktails and save them to the ontology through Owlready2 and the user interface, but this would be misleading and allergen information would be overlooked until the reasoned ontology is exported from the Protege interface and integrated into the gui code.

If truly exceptional maintainability were to be achieved, the program would have needed to be developed in Java in order to make use of the OWL API, which offers much more flexibility and performance when programmatically interacting with ontologies. The GUI development would have been more challenging, and the cocktail scraper and parser would have been more difficult to develop, but the overall user experience would likely be much better due to shorter loading times and increased maintainability.

Chapter 7

Conclusions and Further Work

7.1 Summary of Results

It is a challenge for restaurants and bars to consistently provide excellent service and hospitality, but tools such as Booze Muse can make this much simpler. The application was perceived as aesthetically pleasing and intuitive to use and represented a bridge between creativity and craft. The main application interface, hosting the allergen search and general cocktail search along with navigation buttons, was simple to use and presented no functional issues to the users; feedback received commented on font and search bar colors.

Users felt that the menu generation tool in particular was an element with a major practical application, and the exported menu and allergen information make this function genuinely useful and applicable. However, more customizability was desired, and informative feedback and continuity were lacking in the user interface.

The advanced query page successfully abstracted the users away from the underlying ontology while maintaining a high degree of flexibility. Users found it intuitive to use, but loading times were too long actions were not reversible, which detracted from the overall user experience.

The cocktail scraper and parser were efficient and accurate and managed to populate the ontology with a wide array of high-quality cocktails. However, the parser did repeatedly misclassify certain ingredients, and this error can be mitigated with a more advanced string matching method. There was one instance of a critical error in classification, when ”walnut bitters” was classified as ”bitters”, missing a tree nut allergen classification, but this was due to a lack of certain ingredients being populated in the ontology, and the parser still made a technically correct classification.

The ontology is populated with enough ingredients to represent a large number of cocktails, and allergens present in ingredients and thus cocktails are correctly identified. However, the Owlready2 API presented challenges when interacting with the ontology, and the resultant data structure had poor maintainability and scalability.

7.2 Reflection of Project Aims

Cocktail Ontology

The final ontology is able to describe cocktails in terms of ingredients, but information about preparation, garnish, and serving style is only included as RDFS comments on the cocktail entity. While this does achieve the aims, these elements are not fully integrated into the ontology. They are enough to provide necessary information to the user, but the user is not able to interact with the ontology based on these criteria, e.g. a user could not filter for ”cocktails that are shaken and served in a martini glass”. The ingredients list is extensive enough to represent a wide range of cocktails, but certain areas could be easily expanded to allow for more accurate cocktail parsing and decrease the number of cocktails discarded by the cocktail scraper. The 14 major allergens are all appropriately represented in the ingredients and cocktails that they are present in, but an expansion of the ingredients list would allow for more accurate cocktail parsing and would make the allergen information stored about cocktails more accurate. The knowledge stored about cocktails and ingredients is only maintainable through the Protege user interface, and because of the challenges presented by the Owlready2 API, even the simple and likely common act of introducing a new cocktail to the application is a complex process, vastly degrading the level of maintainability.

Web Scraping and Parsing

The web scraper is able to successfully scrape drinks from the Punch Drink cocktail archives, but the number of ingredients limits the number of cocktails that are successfully scraped. Cocktails are converted to their respective representation in the ontology with a high level of accuracy. The parser does well at discarding cocktails that it does not have the correct ingredients for, but sometimes more general class labels are assigned to ingredients which detract from the informativeness of the cocktail in the ontology.

User Interface

Advanced query operations were simple and intuitive through the user interface, requiring no prior experience and minimal technical proficiency. The underlying data structures were fully hidden from users. The cocktail search function worked as intended, providing users with the cocktail along with its ingredients, ingredients specification, preparation method, glassware, garnish, and allergy information. The allergen search feature allowed users to find cocktails omitting certain allergens, but the presentation of these cocktails could be improved and the loading times were too long. The menu generation feature was able to generate consistently new menus that adhered to industry-standard menu writing guidelines. The advanced query functionality was implemented, but the poor scalability of the application prevents arbitrary levels of specification. Additionally, the UI for ingredient selection was poorly implemented and detracted from usability. The application is able to export menus and their accompanying allergen matrices. However, the menus lacked customizability and could be malformed depending on the cocktails included in the menu. Additionally, the quality of the allergen matrix varied depending on the menu and was unreadable in some instances. A perfectly formed allergen matrix was overwhelming to users and readability was poor.

7.3 Areas of Improvement and Further Works

One critical area of improvement is the scalability of the application. In order for Booze Muse to be practically deployed, the knowledge base needs to include many more cocktails than its current state, and already the application struggles with loading times directly associated with the number of cocktails held in the ontology. One way to improve this performance would be to use another form of data representation for the cocktails and the ingredients. A graph database tool like Neo4J would not provide the same structure as an ontology, would not include a reasoner, and would not include valuable hierarchical data and insights. However, much of the functionality for the user would be preserved if a tool like this were to be used, and because the graph could be queried natively in Python with efficient APIs rather than through the RDF translation layer that is required for Booze Muse as it currently stands, the efficiency would likely be drastically increased, reducing weight times and increasing scalability. Tools like Neo4J are built for real-time interaction, and this would help create a snappier user interface for a much better user experience.

The application as it stands has many dependencies, and while a deployed version would be streamlined to minimize the amount of work for the user to install and run the application, another approach would be to host the application on the web. Hosting the application on a Flask web server would require much more back-end work, but would make accessing and running the application much simpler for the user. Users could start from a base ontology which is maintained by the program administrators and updated with new ingredients, relationships, and so on, while end users could add their own cocktails to their own personal ontology or graph database, hosted on the web, which could be shared between the members of an organization rather than localized to one particular machine. Additionally, if the menus are hosted online already because they are generated through a web application, the process of sharing these menus and integrating them into online ordering systems would be faster and simpler.
Another way that the functionality of the application could be extended would be to increase the scope of knowledge held in the ontology about drinks and their ingredients. For example, fully integrating the information stored about preparation, glassware, and garnish would allow for many more useful search features. Adding properties to ingredients indicating the level of sweetness, bitterness, salinity, acidity, etc would allow for general flavor profiles to be crafted for cocktails, and users would be able to search based on the tastes they are looking for without knowing exactly what ingredients they want in the drink. This additional freedom would make the application more flexible and more approachable for less knowledgeable users that may not know what each ingredient tastes like.
A more complete menu generation tool in which users could add cocktails from their allergen search and advanced query to a custom menu would make this tool much more useful to bar administrators; currently, the application does not allow for a menu to be customized after it has been generated, but it is likely that even if a balanced menu is created by the application, the bar administrator might want to make some adjustments. This simple addition to the application would vastly increase its functionality.

Bibliography

[1] Knowledge graph. Technical report, IBM. https://www.ibm.com/uk-en/topics/knowledge-graph.

[2] Prepacked for direct sale (ppds) allergen labelling changes for restaurants, cafés and pubs.

[3] Selenium, overview. Technical report, Selenium. https://www.selenium.dev/documentation/overview/.

[4] Tkinter documentation. Technical report, tkinter. https://docs.python.org/3/library/tkinter.html.

[5] OWL 2 web ontology language document overview (second edition). W3C recommendation, W3C, Dec. 2012. https://www.w3.org/TR/2012/REC-owl2-overview-20121211/.

[6] A. Arwan, M. Sidiq, B. Priyambadha, H. Kristianto, and R. Sarno. Ontology and semantic matching for diabetic food recommendations. In *2013 International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 170–175, 2013.

[7] M. Bamboat, A. Hafeez, and A. Wagan. Performance of rdf library of java, c and python on large rdf models [scopus index y category journal]. pages 25–30, 01 2021.

[8] D. Baorto, L. Li, and J. J. Cimino. Practical experience with the maintenance and auditing of a large medical ontology. *Journal of biomedical informatics*, 42(3):494–503, 2009.

[9] J. Barnett, N. Botting, M. H. Gowland, and J. S. Lucas. The strategies that peanut and nut-allergic consumers employ to remain safe when travelling abroad. *Clinical and Translational Allergy*, 2(1):12, 2012.

[10] D. A. Booth. Salty, bitter, sweet and sour survive unscathed. *Behavioral and Brain Sciences*, 31(1):76–77, 2008.

[11] J. Bucanek. Model-view-controller pattern. *Learn Objective-C for Java Developers*, pages 353–402, 2009.

[12] A. Day, N. Fauchald, D. Kaplan, D. Tarby, D. J. Ho, J. Afuso, and T. Tomkinson. *Cocktail Codex Fundamentals, formulas, evolutions*. Ten Speed Press, 2018.

[13] T. S. P. de Souza, R. F. Miyahira, J. R. V. Matheus, T. B. de Brito Nogueira, C. Maragoni-Santos, F. F. C. Barros, A. E. C. Antunes, and A. E. C. Fai. Food services in times of uncertainty: Remodeling operations, changing trends, and looking into perspectives after the covid-19 pandemic. *Trends in Food Science & Technology*, 2022.

[14] M. DeBellis. *A Practical Guide to Building OWL Ontologies (Edition 3.2)*, Oct. 2021.

[15] B. Endres, R. Endres, and M. K. Nižić. Restaurant disclosure of food allergens: Analysis and economic implications. *Tourism and Hospitality Research*, 21(2):202–215, 2020.

[16] M. Gowland and M. Walker. The forensic implications of food hypersensitivity–a review of cases in united kingdom courts: January 2014–february 2020. *Perspectives in Public Health*, 142(6):347–354, 2022.

[17] G. Gronier and A. Baudet. Does progress bars' behavior influence the user experience in human-computer interaction? *Psychol Cogn Sci Open Journal*, 5:6–13, 2019.

[18] T. Gruber. *Ontology*, pages 1–3. Springer New York, New York, NY, 2016.

[19] S. Han and C. K. Anderson. Web scraping for hospitality research: Overview, opportunities, and implications. *Cornell Hospitality Quarterly*, 62(1):89–104, 2021.

[20] R. J. Harrington, S. G. Fauser, M. C. Ottenbacher, and A. Kruse. Key information sources impacting michelin restaurant choice. *Journal of Foodservice Business Research*, 16(3):219–234, 2013.

[21] M. W. Jaspers, T. Steen, C. Van Den Bos, and M. Geenen. The think aloud method: a guide to user interface design. *International journal of medical informatics*, 73(11-12):781–795, 2004.

[22] L. Jean-Baptiste and L. Jean-Baptiste. Creating and modifying ontologies in python. *Ontologies with Python: Programming OWL 2.0 Ontologies with Python and Owlready2*, pages 113–133, 2021.

[23] T. Jo. Text mining. *Studies in Big Data*, 2019.

[24] M. Lutz. *Programming python: powerful object-oriented programming*. " O'Reilly Media, Inc.", 2010.

[25] D. Meyer. *Setting the table*. HarperCollins e-Books, 2014.

[26] A. Norta, R. Yangarber, and L. Carlson. Utility evaluation of tools for collaborative development and maintenance of ontologies. In *2010 14th IEEE International Enterprise Distributed Object Computing Conference Workshops*, pages 207–214, 2010.

[27] S. Perumal and S. Rawal. Handling cold-start problem in restaurant recommender system using ontology. In *Proceedings of Emerging Trends and Technologies on Intelligent Systems: ETTIS 2022*, pages 319–329. Springer, 2022.

[28] G. Popovski, S. Kochev, B. Seljak, and T. Eftimov. Foodie: A rule-based named-entity recognition method for food information extraction. *Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods*, 2019.

[29] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C recommendation, W3C, Jan. 2008. https://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[30] B. Shneiderman. Shneiderman's eight golden rules of interface design. *Retrieved july*, 25:2009, 2005.

[31] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

[32] A. Sonderegger and J. Sauer. The influence of design aesthetics in usability testing: Effects on user performance and perceived usability. *Applied Ergonomics*, 41(3):403–410, 2010. Special Section: Recycling centres and waste handling – a workplace for employees and users.

[33] V. Svátek, O. Šváb-Zamazal, and V. Presutti. Ontology naming pattern sauce for (human and computer) gourmets. In *Workshop on Ontology Patterns*, volume 171, 2009.