1.1 Introduction

This document presents the architecture and detailed design involving the software developed for the PureDine project. This project performs as a web application that can be used to search up local restaurants with their menus to check for different types of allergies possible on them.

1.1.1 System Objectives

The objective of this application is to provide an interface to provide users, either signed in through Firebase integration or standalone, the ability to search for local restaurants in ways including, but not limited to, specific name, city, or zip code. Once searched, the software will list either specific or local restaurants in a local area range each defined with a separate box containing information about them. These boxes can be clicked on to show basic menus of the restaurants with checkboxes at the bottom to mark certain allergens. Once marked, the website will analyze items on the menu with their ingredients to give recommendations on what is safe to eat and what could possibly be harmful to a consumer.

1.1.2 Hardware, Software, and Human Interfaces

The following Hardware, Software, and Human Interfaces are used in PureDine design and development.

1.1.2.1 Hardware

The hardware used for the PureDine project can be summarized as follows. The current developer of PureDine develops on a Windows 10 interface presented on a Dell laptop. The laptop itself can be linked to both home and LMU servers to allow for development in different

locations. The laptop comes equipped with a keyboard and trackpad to allow for interaction with the machine but can also support plugins like external mice or keyboards if needed. Presentation of the project will be done on external devices such as projectors provided by LMU.

1.1.2.2 Software

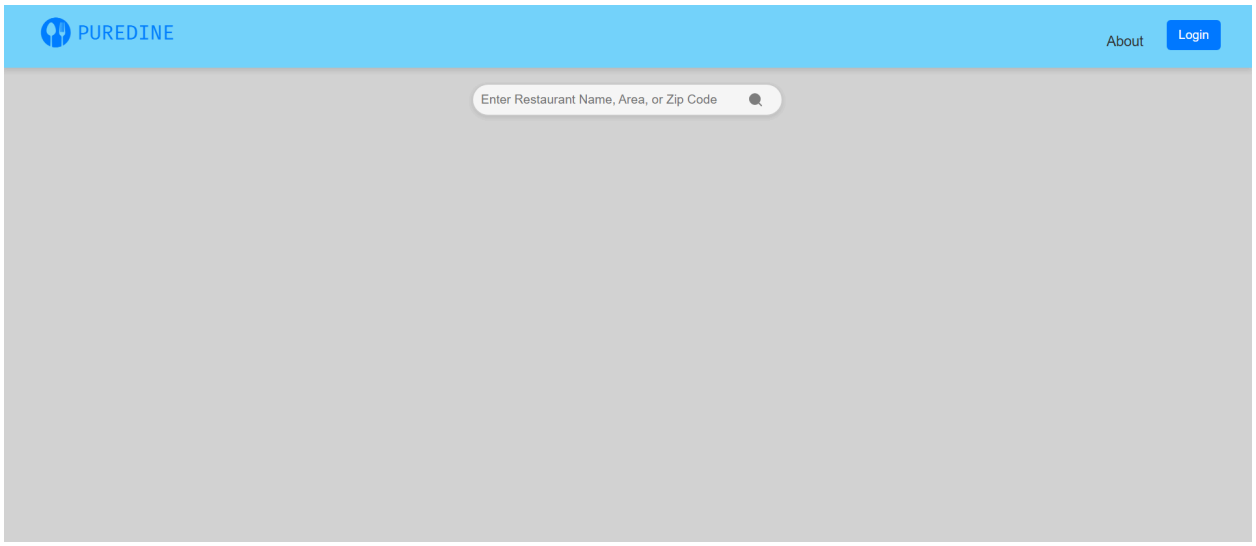
The software used for the PureDine project can be summarized as follows. External browsers such as Safari, Google Chrome, Opera, and Firefox will be used to host the broadcast of the website to users when used. Visual Studio Code will be used to interact with and modify files for development use. React is used for development purposes in association with Node.js to work on frontend and backend processes of the PureDine system and its associated applications. APIs such as the Google Businesses API, Possible MenuPortal API, and Edamam API are used for development of certain pieces of the project including restaurant searching, menu sharing, and allergen information and analysis with menu items. Firebase will be used primarily for hosting of the websites itself, but will also be used for user profiles in tuning of specific allergies involving frequent use of the account.

1.1.2.3 User Interface

The UI for the PureDine project can be summarized as follows. The UI of PureDine is split into four main pages: the main homepage, the about page, the restaurant listing page, and the menu / allergen page. These pages will be described in the following paragraphs.

1.1.2.3.1 Main Homepage

The main homepage of PureDine will display the following when first seen. When loaded, the website will show a header containing the following objects: The logo of PureDine that can be clicked on to reload the page, the link to the “About Page”, and a login button for personal accounts. In the center of the page, there will be a search bar with the following text within it “Please enter a specific restaurant, city, or zipcode”. The user will be able to type in this search bar to go to the “Restaurant Listing Page” when entered. There may also be images and other graphics or text around the website to improve presentation and overall style.



1.1.2.3.2 About Page

The about page of PureDine will display the following when first seen. When loading, the website will have the header from the first page with the following objects still present: a back button for returning to the main webpage, and the logo of the site in the center of the header. The image of the logo can also be clicked onto return to the main page, similarly to the original

image. The page under the header will contain a description of the website's origin + story and a small instruction guide on how to use the website properly with and without an account. There will also be notices on the bottom informing the user that all information provided may not be 100% accurate.

1.1.2.3.1 Restaurant Listing Page

The restaurant page of PureDine will display the following when first seen. The page itself can only be accessed through use of the search bar from the home page. Once a location is searched up, restaurants in a local area will display in boxes that go down the page. Either the page will display restaurants with infinite scroll or with several pages containing about 20 or so restaurants. These restaurant boxes will list the following: restaurant name, restaurant type, restaurant hours, restaurant address, and an associated image. Each of these boxes can be interacted with to go to the menu / allergen page.

1.1.2.3.1 Menu / Allergen Page

The menu / allergy page of PureDine will display the following when first seen. Once a restaurant box from the restaurant page is selected, the menu of the restaurant will be displayed generated from the MealMe API. The menu will be listed in a strong format aiming for the look of a typical restaurant menu to be achieved. Under this menu, there will be several checkboxes denoting allergies, including but not limited to: Nuts, Soy, Shellfish, and more. These boxes can be checked and, in association with the Edamam API, the menu will highlight specific items in specific colors with a guide to the right of the screen highlighting if something is safe to eat or

dangerous based on selection. This page will also include the message from the about page where it highlights that everything given may not be 100% accurate.

1.2 Architectural Design

The Architectural Design of PureDine will be broken down into multiple different subsystems. These include the UI Control, API Control, and Database Control described as the following:

UI Control:

The UI for PureDine is designed based on the main App.js file that structures the main page and links to other webpages. The main class is based in classes and subclasses to deal with and handle different parts of the design. In App.js in particular, there are classes dealing specifically with the header and the search bar of the site allowing them to function and interact with the main page. Using the 'react router dom' extension, the site can also link to different pages through different files and functions. These files include the about page (About.js), restaurant listing page (Restaurant.js), and the menu / allergy page (Menu.js). Each of these files contains separate classes and functions to host UI for those separate pages including features such as restaurant listings, menu display, and information on the general site. These files mainly interact with each other through the use of UseEffect functions that allow for site operations and modifications to happen in real time.

API Control:

The API's of PureDine are designed to work based on a main file containing the APIs themselves labeled Api.js. This file contains constants labeled "getRestaurants", "getMenus",

and “getAllergies” specifically which are async functions that contain the API keys for each of the APIs used (Google Places, MealMe, Edamam) along with urls to specific parts of the sites to draw information from. These work in parallel with a constant named response that fetches data from the url and throws an error if no information is collected. Finally, each function has a constant labeled json that collects and returns the information from the site if found. At the bottom of this file, there is an export that exports all of the main functions to allow them to be utilized in other files. Other uses for these functions can include using the search bar on the home page, using the restaurant listing information on the restaurant page, and using the menu selection and allergy tailoring of the menu / allergy page. These API keys are soon to be hidden with environmental variables for safety purposes.

Database Control:

The database of PureDine is designed based on the main firebase_config.js file that allows for connection to the firebase database and allows specifically for the initialization, hosting, and authentication of the project. These constraints are set up as objects which are used throughout the files. Hosting can be done in the back end near the end of the project and can be updated regularly through the terminal. Authentication will use functions such as SignIn, SignOut, and useAuthentication to allow for interaction with objects such as the login and logout buttons.

1.2.1 Major Components

Major subsystems present in PureDine that correlate to the functional requirements of the software present in the Requirements Specification document can be described in the following:

- From 1.3.1.1-1.3.1.4, The UI subsystem is used to render the main webpage and interact with the React systems done specifically by the App.js file.
- At 1.3.1.5, the Database Control system will deal with Firebase in its own firebase config file that will handle specific hosting and authentication.
- From 1.3.1.6-1.3.1.15, the UI subsystem will be used for header design, and website design specifically of the main homepage and the about page, allowing for reloading and access of the two pages thanks to the 'react-router-dom' extension allowing page switching between the App.js and About.js files.
- From 1.3.1.16-1.3.1.24, the API control subsystem will be used to pull information on restaurant data, nutrition information, and menu information from the three APIs in use (Google Places, MealMe, Edamam) and display them specifically in combination with the UI subsystem in the Restaurant.js and Menu.js files.
- From 1.3.1.25-1.3.1.26, the UI and Database control systems will allow for user login through authentication and specifically the SignIn, SignOut, and useAuthentication functions will be used to host and display changes to the webpage based on the Firebase status.
- From 1.3.2.1-1.3.2.6, the API control subsystem will be utilized to display certain information depending on the search bar and restaurant selected involving location data given and specific restaurants selected (Google places, MealMe). The website will also highlight different allergy recommendations and guidelines based upon what is given by the Edamam API.

- From 1.3.2.7-1.3.2.15, The Database Control systems will allow for hosting due to the pushing of files to Firebase through the use of the terminal and login due to the authentication features built into Firebase extensions.

1.2.2 Major Software Interactions

Major software interactions throughout the PureDine application can be described by the following:

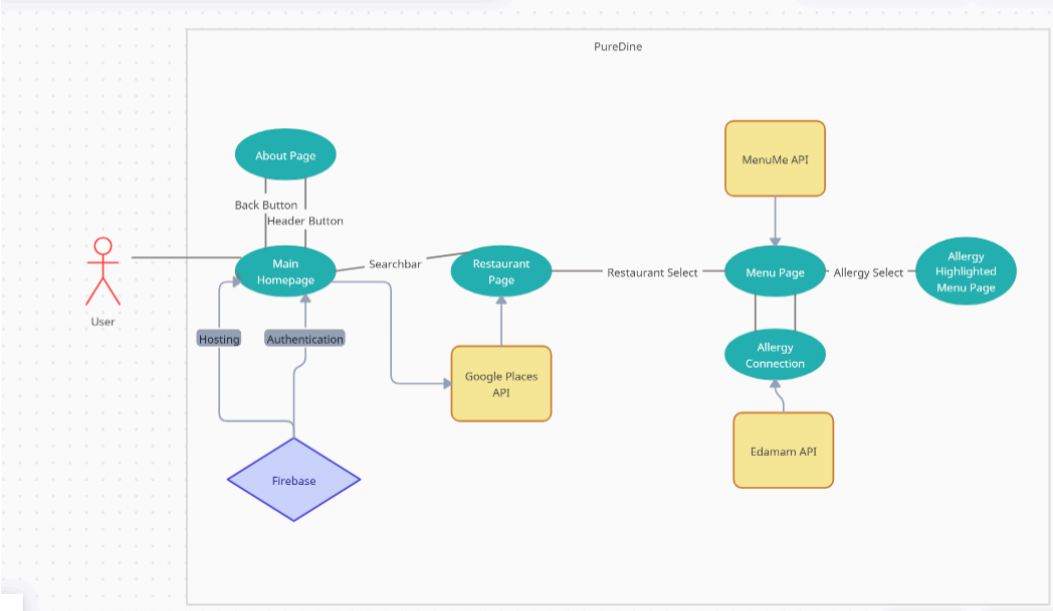
- The main header of the main page can be interacted with to not only reload the main page, but also link to the about page through the use of a link with the text “About the site”. The header will also contain the login button which will allow for a user to sign in or sign out of the site to adjust allergy preferences.
- The search bar of the main page, when given information based on a specific restaurant, city, or zipcode, will interact with the Google Places API to draw information on restaurants in a user's local area and list them specifically on the restaurants page, which the user will be linked to once selections are entered.
- Each restaurant on the display page can be clicked on to give access to the menu / allergy information given by the MealMe and Edamam APIs.
- The restaurants page will list up to 10 restaurants and will either have selections to another page for more listings or will allow for infinite scroll (still undecided).
- The user will have the option to interact with checkboxes on the menu page to select certain allergies or dietary restrictions that may affect them. This will lead to the menu

highlighting specific items with different categories of safety for consumption. These choices can also be made through the use of an account login.

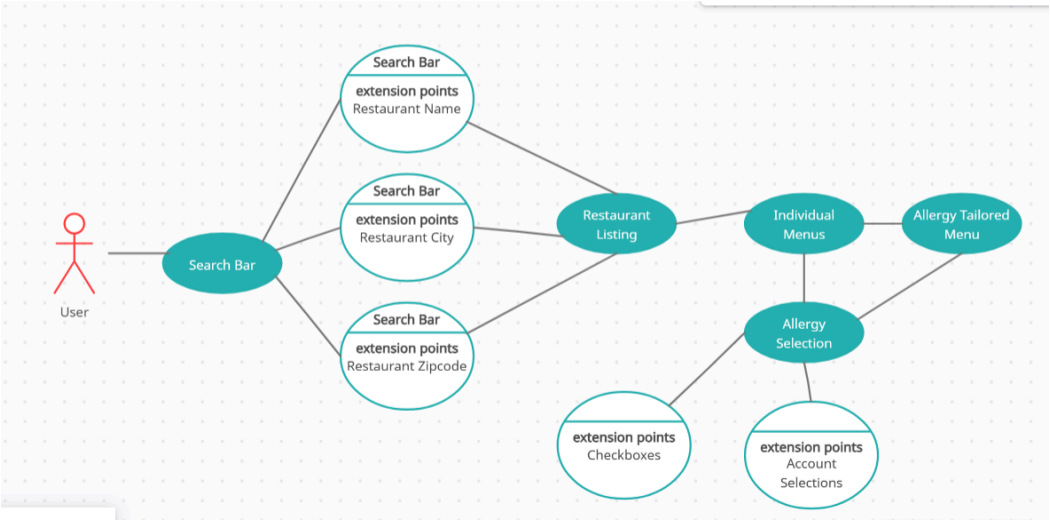
1.2.3 Architectural Design Diagrams

Here are some diagrams for PureDine applications at an architectural level:

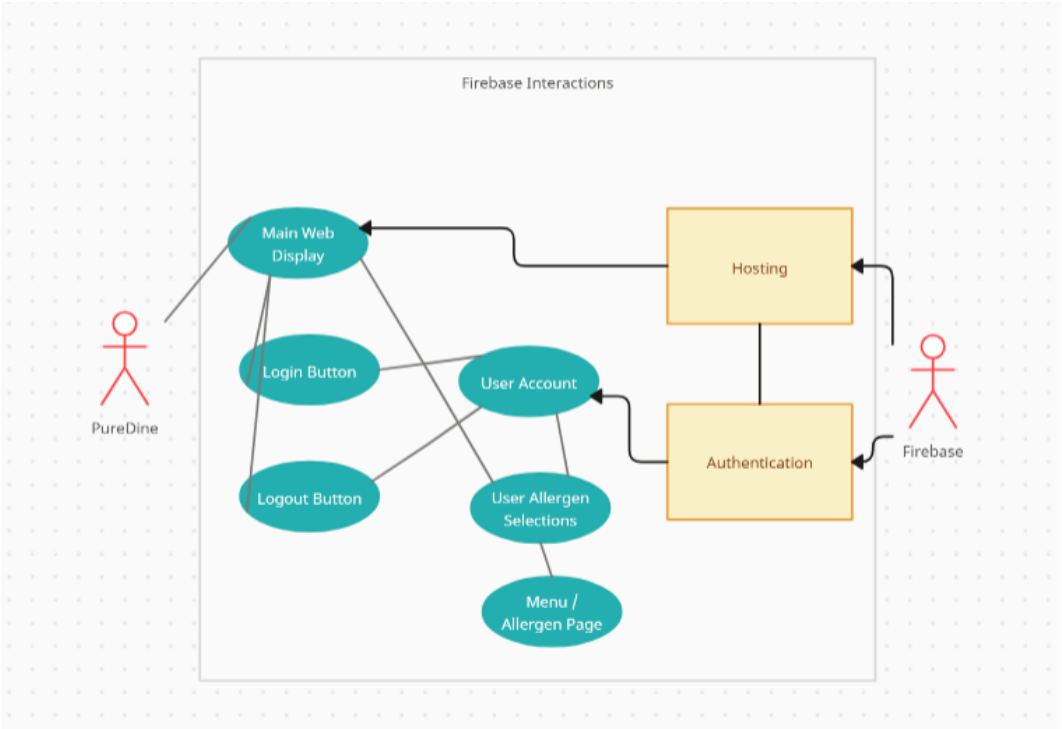
Overall System



Restaurant / Menu Selection



Firebase Interactions



1.3 CSC and CSU Descriptions

The Detailed Design of PureDine will be broken down into multiple CSC's and CSU's in accordance with the system. These will be highlighted down below:

Main App and Header Presentator:

- App
- Header
- Search-Bar / System
- Route System
- Google Sign-In Container

About SitePresentator:

- Return System
- About Description

Restaurant and Menu API Presentator:

- Restaurant Finder
- Fetch Restaurants
- Restaurant List
- Restaurant Card
- Google Places API

Health Summary API Presentator:

- Edamam API

- Health Selection
- Menu Finder
- Menu Lists / Cards
- Modify Menu

1.3.1 Class Descriptions

The following sections provide the details of all the classes used in the PureDine application. These will be split into subsections which will be shown here:

1.3.1.1 App

This class interacts with the main page and all the elements within it. It contains the context to the header class which will be discussed later on, and the route elements to all the other pages. It also contains the Home function that helps with the route functionality and the handleSearch function which allows the search bar to function. This class also contains the signIn and signOut functions along with the functionality for the header Google sign in and out.

1.3.1.2 Header

The header class involves the handling of everything in the main header of the page and information on the general search bar which will be explained later on. The header contains the logo-container subclass which allows for interaction with the PureDine logo in the top left to reset the overall page. There is also a small nav-links class that allows for routing to the about page and the home page. There is also the login button working with an onClick function to

allow for Firebase interaction. This is the class that also uses the imports from the react router-dom system and useState from React to the highest extent.

1.3.1.3 Search Bar / System

The search bar system contains the handleSearch function mentioned previously and the values to create and interact with the search bar itself. The handleSearch function trims each search input to allow it to be more functional with the Google Place API and uses a constant labeled navigate to query the restaurants based on zip code, city, and individual name. The search bar itself is made of a search bar subclass that has the text “Enter restaurant name, city, or zip code” as a search input. This search bar can be added to with an onChange function and, once enter is pressed, loads the Restaurant page for future access. There is also an icon associated with this search bar which is created through width, height, and fill values seen in the html code.

1.3.1.4 Route System

The route system of PureDine is done through the react-router-dom import which contains a Route function that contains navigational links to the home page, about page, the restaurant page, and the menu page. This is done by defining a path in the files as well as a name for the page that the path will lead to with an element key. There is also a small BrowserRouter import to allow for the switching of website pages as the website is interacted with through the Layout keyword.

1.3.1.5 Google Sign In Container

This class has the sole function of displaying the Firebase Google sign in the submenu when the login button is clicked in the header of the main App page. This is done with the onClick

function from React and the signIn and signOut functions defined in the same file. This function in particular is a try catch handleSignIn constant that awaits the sign in pop up by calling the GoogleAuthProvider function Firebase presents. . If there is an error, the console will display the message “Error signing in” and alert with an error message to the site itself. This function then returns the button that allows for sign in present on the top right of the header titled “Login”. This also works with the similar SignOut function that creates a constant titled handleSignOut which awaits authentication to sign out and, with a try catch, displays an error if failed. Once signed in, the sign out button is provided if a user wants to sign out.

1.3.1.6 Return System

This is a simple class built on the About.js file to allow for navigational return to the main homepage. This is done with a constant titled navigate that uses the useNavigate function in association with react-router-dom import. On the page itself, there is a back button with the text “Back” that, with an onClick function, allows for a user to route their page to the main page and reset the site information.

1.3.1.7 About Description

The about description class is a small class used to contain h1 and p elements in html to write out text explaining the site's history and how it works. This is done in three main paragraphs: an about section explaining the site’s story, a “What is” section explaining how the site itself works, and a note section detailing information on accuracy in the website's code. This is also used with an image import of the author of the site Aidan Esposito with associated css elements. All of this is contained in a giant div section of the site.

1.3.1.8 Restaurant Finder

The restaurant finder function works in combination with three constants (query, restaurants, and loading) to allow for association with the Google Places API. The constant query allows for querying of the restaurant data given in the search bar. The constant restaurants is a useState function from React to allow for data found from the API to be shown. The loading constant works in combination with a useState function to show if the API is fetching data and making changes clear to the user. This class also contains information on a useQuery function that allows for search parameters to be defined and cleaned with the API.

1.3.1.9 Fetch Restaurants

This class contains a useEffect function which contains a constant titled fetchRestaurants which is an async function that checks if a query is valid. This works with a try catch function that sets the loading constant to true, and shows the restaurant data if found. If there is an error, the console will display an error with the text “Error fetching restaurants” in association with the query element. Finally, the setLoading function is set to false to stop the loading text from being shown. The fetchRestaurants function is called at the bottom of the class with the associated query to allow the class to work in the first place.

1.3.1.10 Restaurant List

This class is made to display the general search results from the query from Fetch Restaurants. The page, once a query is found, will display the text “Search results for…” with the associated

query name. This information is then checked with the loading constant to display loading text if applicable. This class is also built with a check that makes sure the amount of information displayed is greater than zero before displaying the information in the ResturantCard class. If information is not found, the site will simply display “No restaurant found” before exporting the restaurant information.

1.3.1.11 Restaurant Card

The Restaurant Card class is a simple class that displays the restaurant information associated with a place when found by the Restaurant List class. This class is contained in a <ul> header that checks the restaurant information given and, using an array, shows restaurant information contained in different categories. These categories contain the restaurant id associated with the location, the restaurant name, the restaurant address, and the general rating of the restaurant in stars found on Google. If any of this information is not found, the text “N/A” will be shown on the screen.

1.3.1.12 Google Places API

The Google Places API class works in association with the getRestaurants constant function which works to get API information. This is an async function with a query parameter that works with another constant known as url that displays a path on where to fetch data from the Google Places API. Another constant known as response is an await function that waits for the data to be collected. If a response is not found, an HTTP error will be thrown to the console while also displaying the response status. If a response is found, the response.json information will be returned and awaited to be displayed. This function is then exported at the end of the Api.js file.

1.3.1.13 Edamam API

The Edamam API class works almost the exact same as the Google Places API class. The class involves a function titled “getAllergy” with a url constant with a different path than the Google API to fetch information. It also has the response constant and the response check to make sure data is transmitted properly. The response is then returned as a json package similarly to the previous function. The function is then exported at the end of the Api.js file.

1.3.1.14 Health Selection

The health selection is a small class made in the Menus.js file that works in association with the getAllergies function as an import to this file. The import is used in combination with a section that contains checkboxes with associated text listing them as a certain allergy. These work with an onClick function to allow these boxes to be interactive and, once selected, work with the getAllergies function to sort the menu data provided from the restaurants. This class also contains the ability to select different highlight colors to the menu text confirming if it's safe, not safe, or risky once a box is selected. Errors can be thrown doing this with the try catch function it is in, making sure the right color is thrown out at the right time.

1.3.1.15 Menu Finder

The menu finder class works very similarly to the restaurant finder class which displays the results from the previously mentioned getRestaurants function. It has a fetchMenus function that finds the options on a menu with its associated css as found via a restaurant from the Google Places API. This information is only shown when a restaurant is clicked on from the Restaurants page. This is an async function that checks if the query provided is valid. It then works with a try catch function to check if the data is found and throw an error if no menu data is present. There is also a setLoading function that allows for loading to be shown or not depending on the situation. The function is then exported for use in other files at the bottom of the page.

1.3.1.16 Menu Lists / Cards

The menu list and card works the same way as the restaurant equivalents. It displays the query found from the menu search displayed previously. It has the ability to display the menu item id, name, and associated photograph if provided. If any of this information is not found, the text “N/A” will be shown on the screen. An error will also be thrown if the searching query can’t find any menu item information.

1.3.1.17 Modify Menu

Modify menu is a small class that works in combination with the health selection class to allow for the highlighting of the information provided in the menu cards. This is done through a function titled useHighlight to allow for highlighting of the different color varieties once information from the Edamam API is provided and queried depending on the allergies provided

in the checkboxes. With a combination of the other class, the menu items can be highlighted and the calculations can be reset if a new checkbox is selected or unselected.

1.3.2 Interface Descriptions

The following sections provide the details of all the interfaces used in the PureDine application. These will be split into subsections which will be shown here:

App and Header Presentator

- **React (UseState):**

Allows for actions to happen across the App, Restaurant, and Menu files (among others). Is used in instances such as button clicking and selection, restaurant selection in the search bar, and checking boxes for allergies on the menu page.

- **Home**

The home component is used as a main app code section working with the react-router-dom import to set the App.js file as the main page and allow routing through the pages within the systems of imported Routes and BrowserRouter.

- **Layout**

The Layout component contains the simple css and layout functionality of the main app page and its subsequent links. It contains the information on the header, about page, and search bar all linked on App.js

- **signIn**

The SignIn component is a small interaction used for displaying the sign in container that is provided with google firebase.

- **signOut**

The SignOut component is a small interaction used for displaying the sign out container that is provided with google firebase.

- **useAuthentication**

The useAuthentication component works with Firebase in order to check if a user is “subscribed” to the current website. If a user is subscribed, this will update the user data and display certain parts of the website. It also allows a user to “unsubscribe” and log out of a website.

- **useLocation (React-Router-Dom)**

The useLocation component allows for a return in the current navigation action which can show the pages current location via a replacement in the history stack. This can be used to set the current page as either the restaurant, menu, or about page from the main App.js file

About Presentator

- **useNavigate (React-Router-Dom)**

This component returns the parent route of the website page hierarchy allowing for a website to return to the main page (App.js) if something is interacted with such as an onClick() to do so. This is specifically seen in About.js with the back button.

- **About**

Contains all the information about the general page including its text, imports, images, and other features with onClick() such as the pages return button.

Restaurant Presentator

- **React (useEffect, useState)**

The UseEffect component is an Interaction imported from React that allows for the use of effects or interactions throughout the site. This can allow for elements of the project to update over the site's use. This can include displaying the restaurants and their changing attributes given from the Google Places API and the Edamam API selections from the health menu.

- **useLocation (React-Router-Dom)**

This element was described earlier in documentation.

- **getRestaurants**

The component getRestaurants works via being an imported library and is used to pull API data from the Google Places API via html calls to a specific route to be used in the restaurants and menu pages of the website. This setup works either with a personal setup network (Server.js) or an online version of the launched website on Firebase.

- **Restaurants**

The restaurants component contains all of the data on the restaurants given from the Google Places API, pulls from the data to specifically find pieces of information such as the restaurant name, address, star rating, and photo, and allows them to display with proper css on the restaurants page. This enables a proper API path between the website and the Google Places API to allow for data to be collected given the conditions of the search bar operation are met.

Menu and Health Presentator

- **React**

The React component is a general instance that brings in all of the features usable from all of React. These include the useState and useEffect instances as well as the general features react offers such as interaction with css and html as well as providing access to their hosting and modification of website services.

- **useLocation (React-Router-Dom)**

This element was described earlier in documentation.

- **getMenus**

The component getMenus works via being an imported library and is used to pull API data from the Google Places API via html calls to a specific route to be used with the menu page of the website. This setup works either with a personal setup network (Server.js) or an online version of the launched website on Firebase.

- **Menus**

The restaurants component contains all of the data on a specific restaurants menu given from the Google Places API, pulls from the data to specifically find pieces of information such as the menu item name, cost, and associated photo and allows them to display with proper css on the menus page. This enables a proper API path between the website and the Google Places API to allow for data to be collected given the conditions of the search bar operation are met.

- **getAllergies**

The component getAllergies works via being an imported library and is used to pull API data from the Edamam API via html calls to a specific route to be used with the menu page of the website. This specifically allows for the scanning of items on the menu page to follow Edamam

given health recommendations and guidelines. This setup works either with a personal setup network (Server.js) or an online version of the launched website on Firebase.

Authorization and Hosting useEffects and Providers

- **Firebase (initializeApp, getAuth):**

The Firebase setup (using initializeApp, getAuth) configures essential Firebase services for the application. initializeApp initializes the Firebase app with the project's credentials and getAuth enables user authentication. This setup allows secure user management and file storage within the app.

- **App (Firebase Config)**

The App component initializes Firebase with project-specific configuration settings. It uses Firebase Config to connect the app to Firebase services, enabling functionalities like authentication, and database. This setup establishes a foundation for Firebase-powered features throughout the application.

1.3.3 Data Structure Descriptions

The following sections provide the details of all the data structures used in the PureDine application. These will be split into subsections which will be shown here:

- **Arrays:**

In the PureDine application, arrays play a crucial role in managing and storing data through the site, specifically those given in user health selections and in API data found from html calls. Specifically, arrays such as restaurant list and menu list allow for the storage of attributes given to a specific restaurant or menu in the system and allows them to be taken and displayed on the proper channels. Arrays also help in the selection of user profile health selections, allowing a user's data to be saved to an array associated with them that can be picked from automatically when looking at items on the menu page.

- **Objects:**

In the PureDine application, objects are a fundamental data structure used to represent and manage the properties of various elements on the website itself. The header contains different links and pictures associated with objects to allow for a concise and easy way to access data and links related to other webpages. The website's search parameters such as query and loading are also used in relation to objects in order to capture user interactions with the system and allow for website changes in operations in real time.

- **Booleans:**

In the PureDine application, booleans are a major component used to check for errors throughout our systems. These can specifically be seen when throwing out errors when fetching data from the API’s, allowing for the user to understand why their data was not collected properly. Booleans are also used to check if every piece of specific data is collected through an API to

make sure everything given can display properly and, if not given, replace the given data with the word “N/A”. Booleans are also used with the button logic to ensure proper routing between the webpages and, if not working, gives the user a way to reset the site and allow them to start from scratch, preventing crashing.

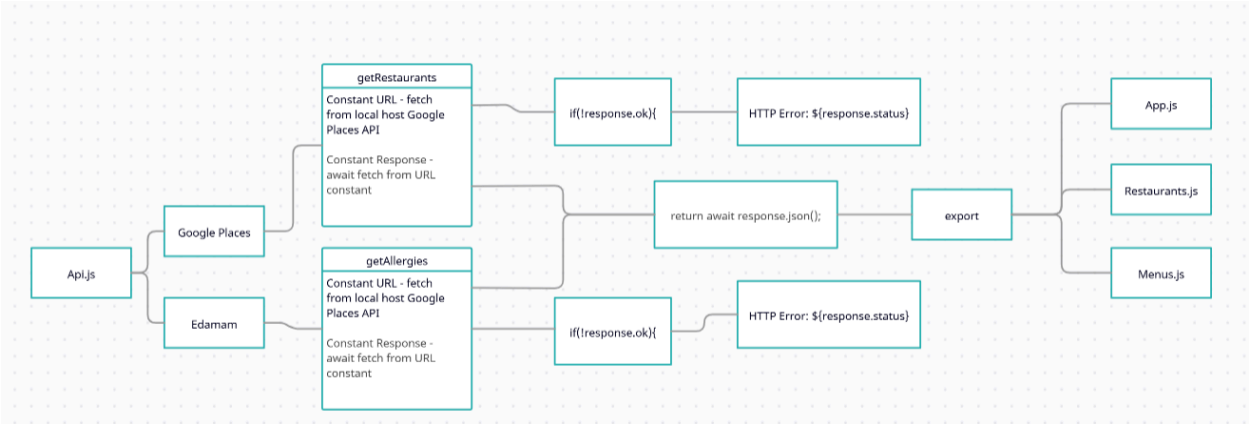
- **Queries**

Search Queries is probably the most important data structure when it comes to collecting and filtering data. Without the filtering and specific searching of data, the data on a restaurant or menu would overload the website with information, making the page unclear and usable. With search queries, data can be refined and specific to allow for only certain items on certain places to present themselves and, if need be, change to react to the current state of a restaurant or menu.

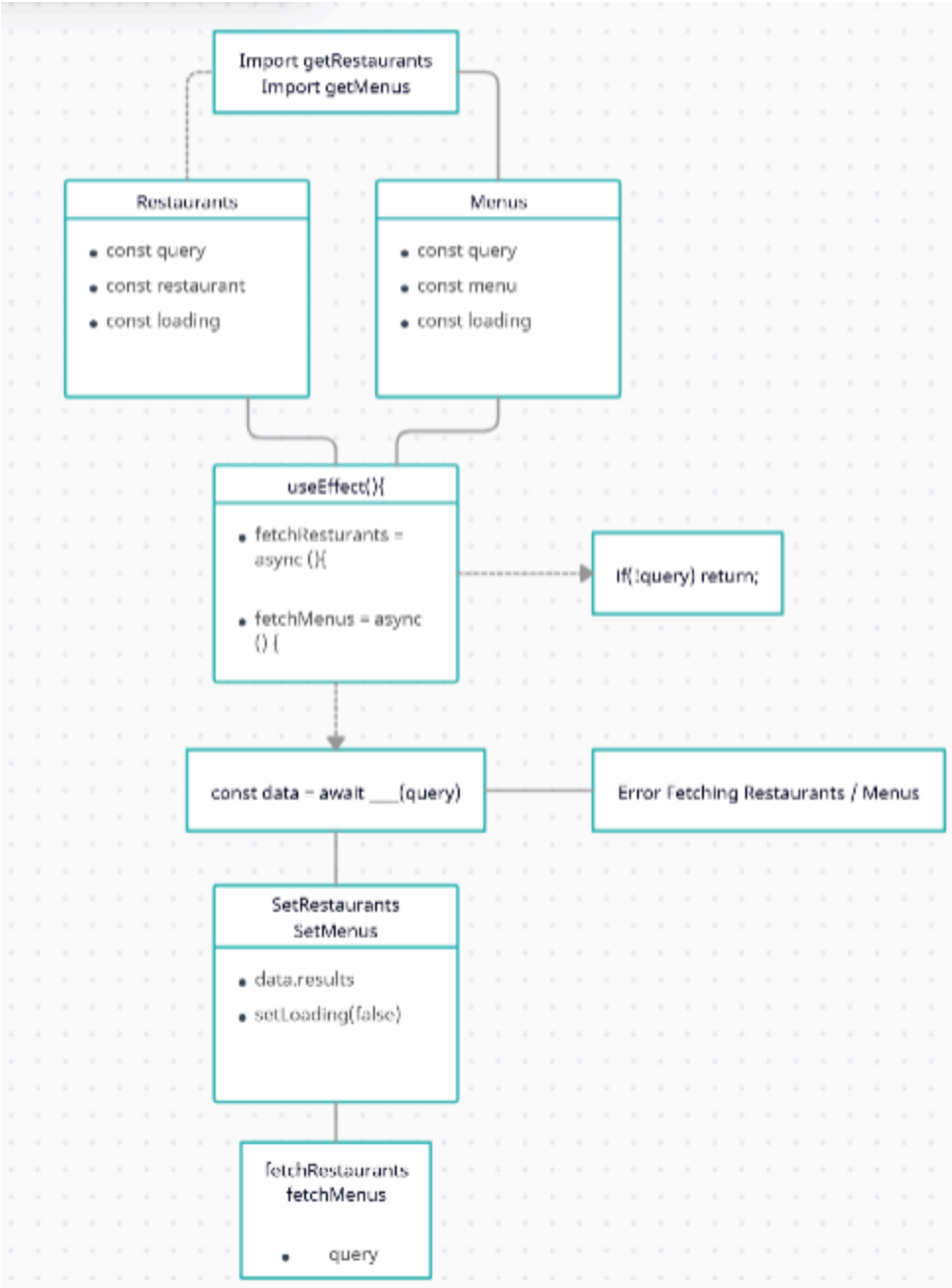
1.3.4 Design Diagrams

Here are some diagrams for PureDine CSCs and CSUs at a design level:

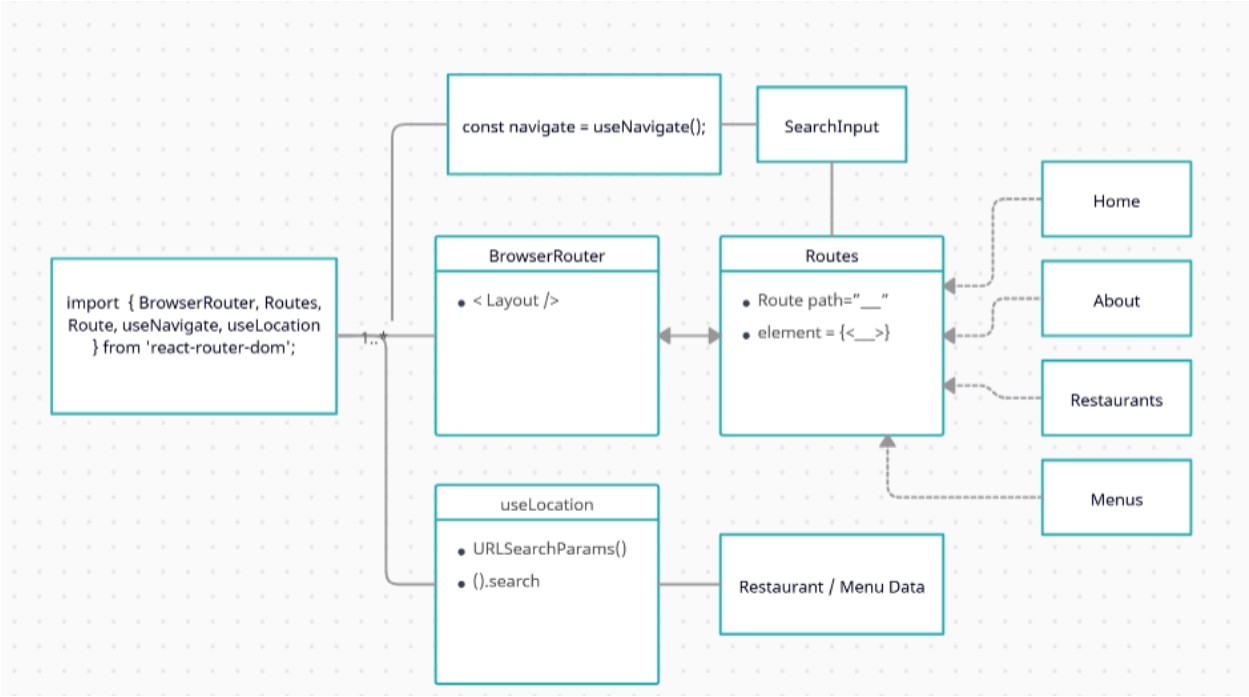
Google Places API + Edamam API



getRestaurant / getMenus Functionality



React-Router-Dom Classes and Routing

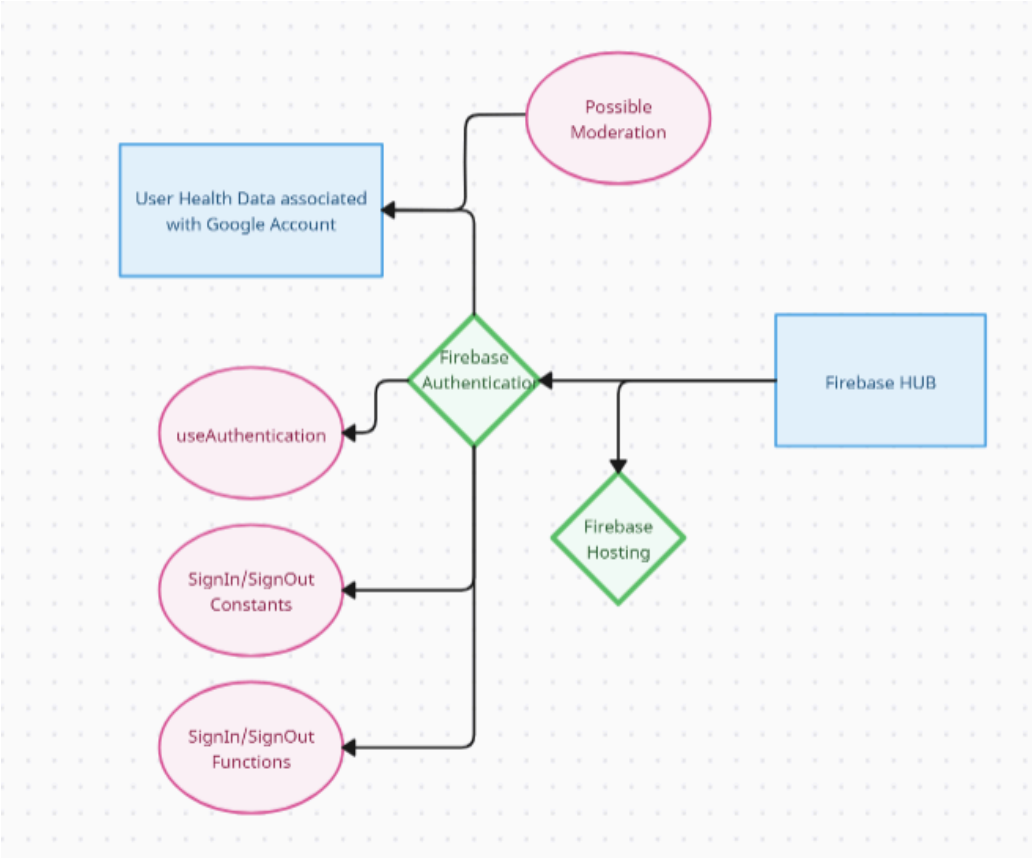


1.4 Database Design and Description

PureDine will use Google Firebase to access database based applications, specifically involving hosting and authentication. The developer of the PureDine website, Aidan Esposito, will have access to the Firebase console. PureDine hosting is done by using the Firebase servers to host the website. The terminal is used to run updates and push changes to the online website. Authentication is currently only used with Google accounts that can be logged into the database, giving information on a user's email and sign in date for information on user selected health conditions and allergies.

1.4.1 Database Design ER Diagram

Here is a diagram describing PureDine's use of Firebase through its systems:



1.4.2 Database Access

The database will be accessed through several avenues throughout the PureDine code including the following:

ConfigFirebase.js:

Pulls from the personal PureDine project website and allows for access of functions in other files such as getAuth, and initializeApp before exporting them as objects titled auth, and app.

App.js

Imports the “useAuthentication” function from ConfigFirebase.js to link to a const titled “user.” Creates references titled “onSignIn” and “onSignOut” that are utilized to sign in and sign out Google accounts and user info.

1.4.3 Database Security

To ensure security from the Firebase platform, Firebase allows for specific rules based for storage and authentication. For authentication, only Google users are allowed to sign into the website. As mentioned, Firebase gives access to the users account information including email and sign in date. Each user also has an associated ID to the website which can be used to limit or remove a user from website access with said account if need be. Users will not be able to access any other users health choices and allergies since all of that information will be associated with authentication on the website backend. On my end, I am looking to establish rules and checks in the code that will prevent users from messing with settings that Firebase cannot block out.