For our website, we decided to make a multi-page website that is based on the MVC model, integrating concepts from both PSET7 and PSET8. To obtain nutritional data and the daily menu, we query the CS50 food API. We used a SQL database to store user information and their associated meal information, utilizing PSET7's query function to query our database. We also extensively used javascript and jquery to dynamically generate html and select DOM elements, and to make API calls to the CS50 food api, respectively.

Several files of our website are incorporated from PSET7, including config.php and functions.php, which include several useful functions such as query, render, and apologize. The rest of our pages rely on a mix of php and javascript. In essence, our overarching approach was that for each of our pages, we use php to generate the bulk of the html that is rendered on each page. To get content from the CS50 food api, we use the $.getJson method of Jquery, which returns the desired information as a JSON object. We then parse the JSON object and generate html on the page using the jquery .html method in order to dynamically generate html on our page that incorporates information received from the CS50 food API.

Specifically, our website has three main features or functions: 1) allow the user to see the menu for the current day, 2) allow the user to create a meal, calculate the nutritional profile of the meal, and store the meal (if they wish), and 3) allow the user to view their history of saved meals.

For the first feature of allowing the user to see the menu for the current day, we chose to display the day's food options for a given meal using an accordion menu. The user is able to use a form with a dropdown to select the meal for a given day. The generated html links in a javascript script, menu.js. Upon clicking the go button, a portion of the script menu.js is triggered which generates a url to query the CS50 food api. We use jQuery to select the meal time from the dropdown div element. We use the jquery $.getJSON method to query the food api and obtain a JSON object which includes the day's food items, as well as their categories. Using javascript, we generate strings of html for each of the food items and package them as paragraph elements. We then use the .html() method to update the html within the div element on the menu_form.php page, called cssmenu. Our accordion menu was built entirely using CSS, using code from this resource: http://www.sitepoint.com/css3-vertical-accordion-using-target-selector/. The foods in the accordion menu are grouped based on the food category (eg desserts, or entrees). Our main design decisions in this section included the decision to present the food data as an accordion menu. We decided this menu is the best presentation format, as it allows the user to see all possible categories of food, then click on the separate tabs of the accordion to access more detailed lists. We decided to use jQuery to query the API because it is the simplest way of making a request to the server, and it is also returns objects as JSONS, which are easy to parse and elements inside the object are easy to access. Further, because the method is asynchronous and we dynamically update the html in the div elements of the page using the jquery .html() method, the user doesn't have to reload the page to see the menu. Finally, we chose to use a CSS-only accordion menu because using CSS transitions allowed us to simplify and streamline our code for the accordion menu.

For the second feature of allowing the user to build a meal for a given time, we used the menu functionality as a starting point. Like before, the user utilizes a dropdown menu to select a meal time for the given day. Upon selecting the meal time and clicking the Go! Button, we generate an accordion menu with the meal's food options. Now, we create the accordion menu that includes the day's menu as before, but this time, each food item is also a button such that when clicked, it adds the food to a meal list at the top of the page. We did this by using a jQuery class selector, and using the event.target selector in jQuery. In this way, all of the food items are presented in the DOM as elements of the same class, but we can select the specific div element that the user clicked on. Finally, to prevent users from entering conflicting food items, for example dessert from lunch and an entrée from dinner, we took advantage of the jQuery .hide() function to hide the dropdown and go buttons when the user clicks the go button the first time.

Next, the user can click on another button that submits the selected foods by iterating through all of the list elements using the jQuery .each method and putting the values into an associative array, which is passed to another controller, nutrition.php. nutrition.php, prints out the associative array which includes all of the food items in the meal. However, we ran into a design issue in this case because we can only query the food API using the recipe number. In order to take care of this, we kept from our original meal selection a JSON consisting of keys that are recipe numbers and values that are the corresponding food name and passed it through as well. Furthermore, we also keep track of the meal time (BREAKFAST, LUNCH, DINNER, or BRUNCH) in the same manner, passing it through with the render function or via invisible, static input fields whenever we made POSTs. Our JSON came in handy because then we could display actual food names alongside our nutritional info along with the portion size and nutrition facts.

Next, after the user has the list with the food, portions, and nutritional info, they can choose to save the meal. This "save" button submits an invisible form via POST to meal_to_db.php that echoes the same values shown in the table. This form contains the total nutritional value for each of our categories, as well as a separate input field for each food item and each associated portion. This PHP document packs the food and nutrition data into two JSON strings via simple string handling and then saves it along with user id and the meal time that we have been keeping track of to a SQL table called mealhistory. We chose to use JSON strings because they are easy to encode and decode, and we can easily store large meals with them.

Finally, when the user clicks the button in the menu that gets the history, we go to history.php. This PHP script queries all the meal rows with the users id, and unpacks the two JSONs via json_decode and foreach loops and puts the data into a dropdown box that is labelled with the meal time and date, along with all the foods with their portions and the total nutritional facts. This dropdown format is designed so that the user can easily track, for example, what they had for lunch last Thursday and get all the relevant information.

Eric, Rohil, and Steven

This was Negative 15.