## Name

Alex Benasutti

## VM

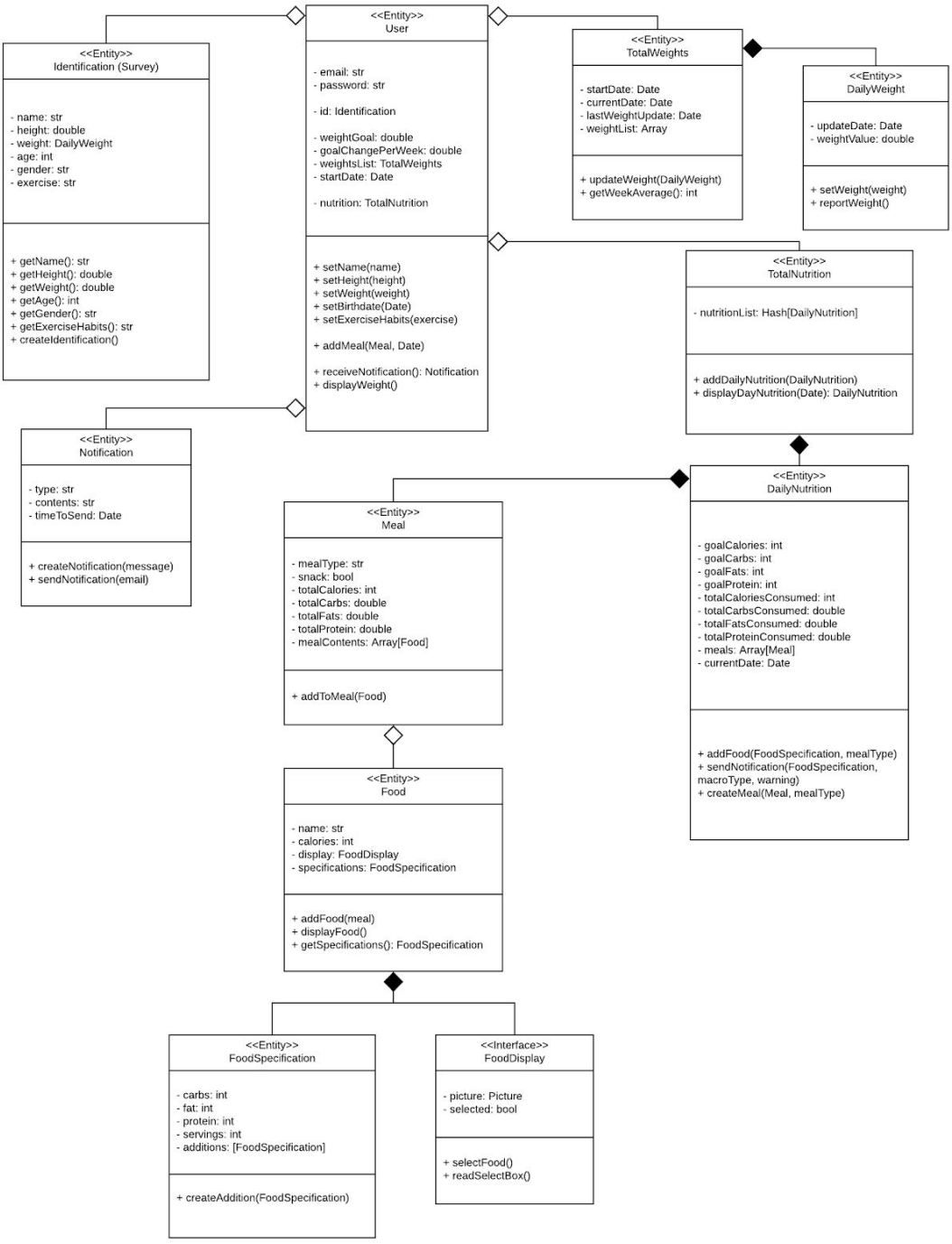
`csc415-server21.hpc.tcnj.edu`

`/home/student1/Nutrify/`

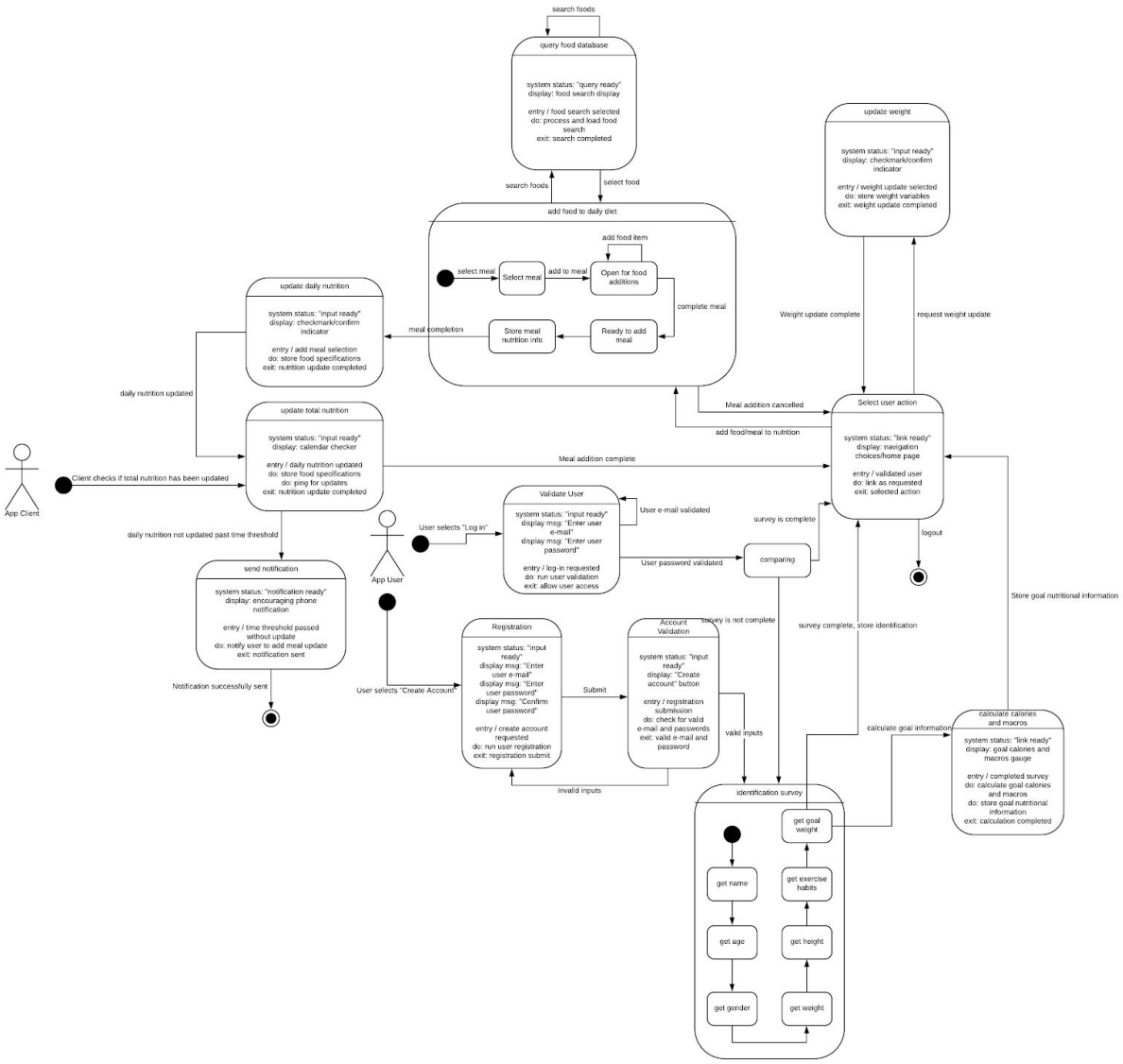
## Github

https://github.com/Jamboii/Nutrify

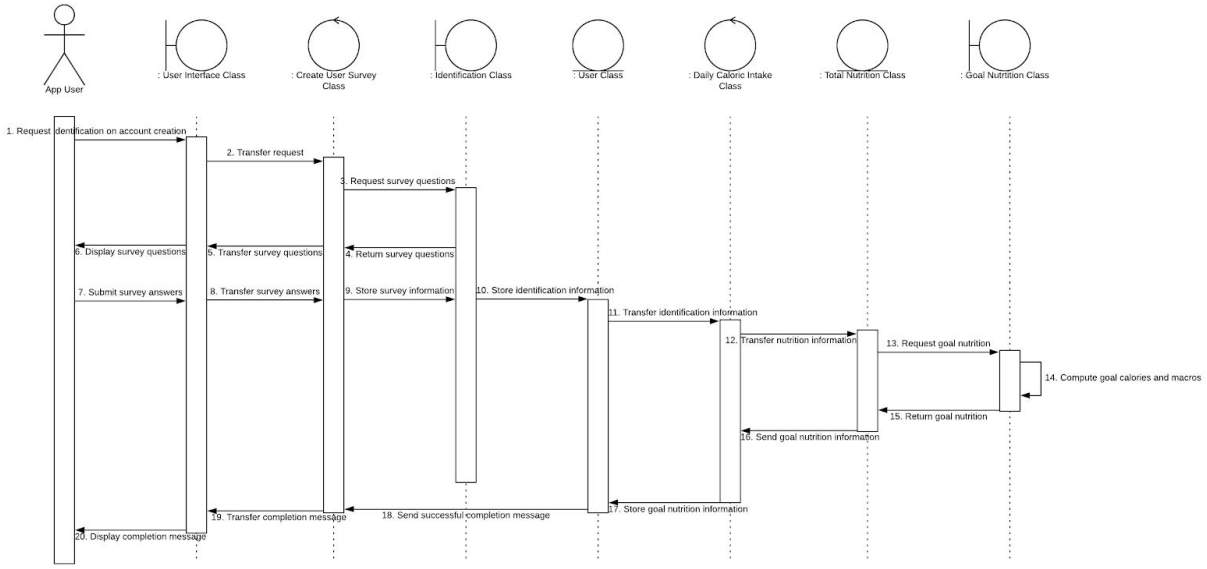
# Design Class Diagram

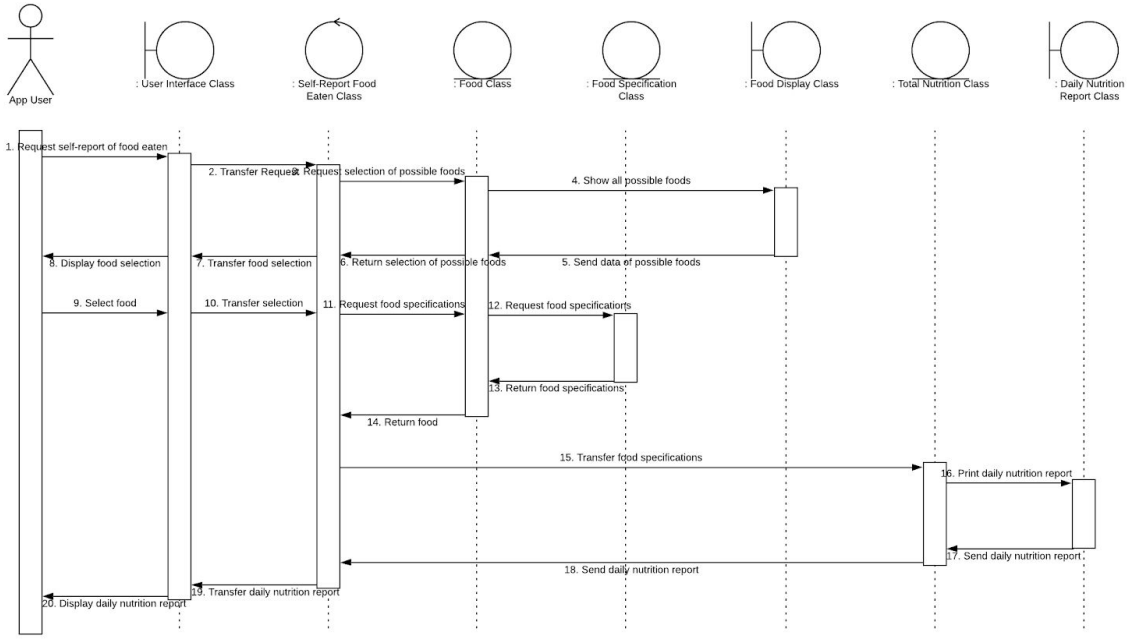


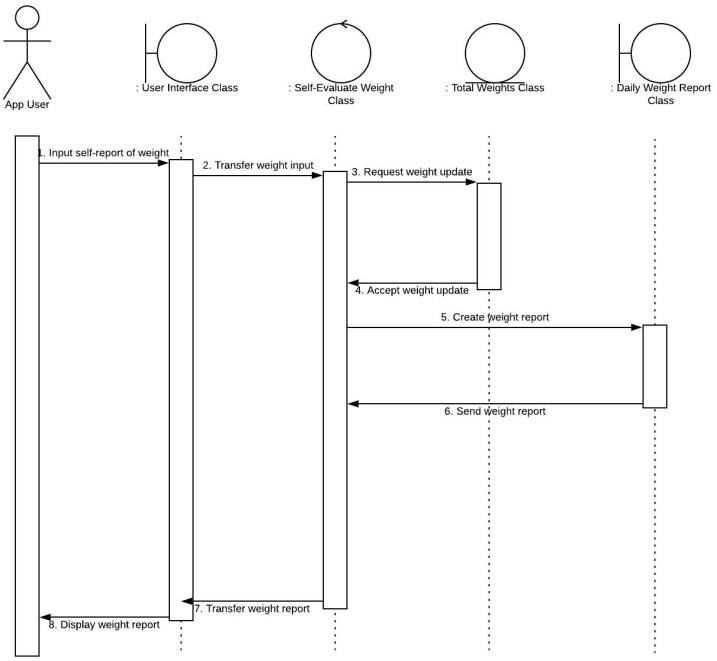
# State Chart

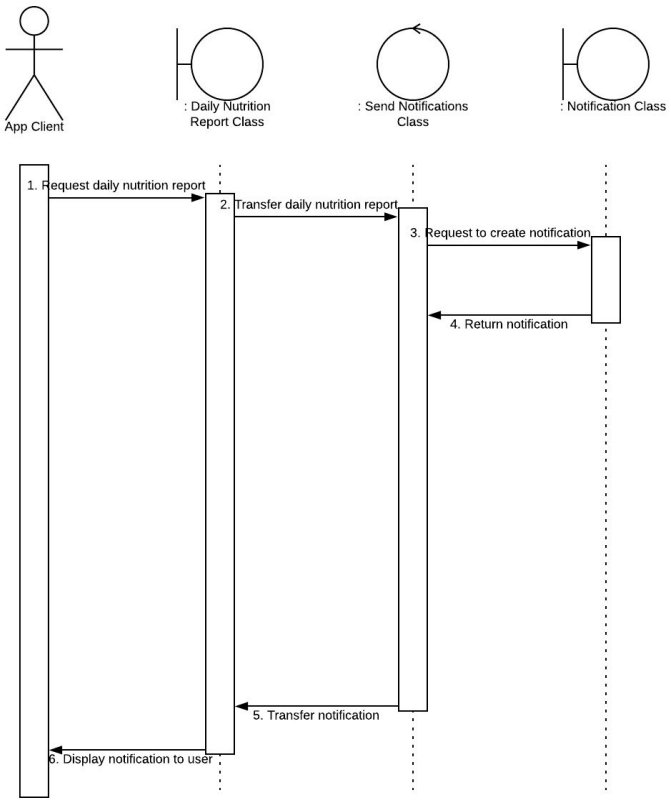

search foods
query food database
system status: "query ready"
display: food search display
entry / food search selected
do: process and load food search
exit: search completed
search foods
select food
add food to daily diet
add food item
select meal
Select meal
add to meal
Open for food additions
complete meal
Store meal nutrition info
Ready to add meal
meal completion
update daily nutrition
system status: "input ready"
display: checkmark/confirm indicator
entry / add meal selection
do: store food specifications
exit: nutrition update completed
update weight
system status: "input ready"
display: checkmark/confirm indicator
entry / weight update selected
do: store weight variables
exit: weight update completed
Weight update complete
request weight update
daily nutrition updated
update total nutrition
system status: "input ready"
display: calendar checker
entry / daily nutrition updated
do: store food specifications
do: ping for updates
exit: nutrition update completed
Meal addition cancelled
add food/meal to nutrition
Select user action
system status: "link ready"
display: navigation choices/home page
entry / validated user
do: link as requested
exit: selected action
Meal addition complete
App Client
Client checks if total nutrition has been updated
Validate User
system status: "input ready"
display msg: "Enter user e-mail"
display msg: "Enter user password"
entry / log-in requested
do: run user validation
exit: allow user access
User e-mail validated
User password validated
comparing
survey is complete
logout
User selects "Log in"
App User
daily nutrition not updated past time threshold
send notification
system status: "notification ready"
display: encouraging phone notification
entry / time threshold passed without update
do: notify user to add meal update
exit: notification sent
Notification successfully sent
Store goal nutritional information
survey is not complete
survey complete, store identification
Registration
system status: "input ready"
display msg: "Enter user e-mail"
display msg: "Enter user password"
display msg: "Confirm user password"
entry / create account requested
do: run user registration
exit: registration submit
User selects "Create Account"
Submit
Account Validation
system status: "input ready"
display: "Create account" button
entry / registration submission
do: check for valid e-mail and passwords
exit: valid e-mail and password
valid inputs
Invalid inputs
calculate goal information
calculate calories and macros
system status: "link ready"
display: goal calories and macros gauge
entry / completed survey
do: calculate goal calories and macros
do: store goal nutritional information
exit: calculation completed
identification survey
get goal weight
get name
get exercise habits
get age
get height
get gender
get weight

# System Sequence Diagrams



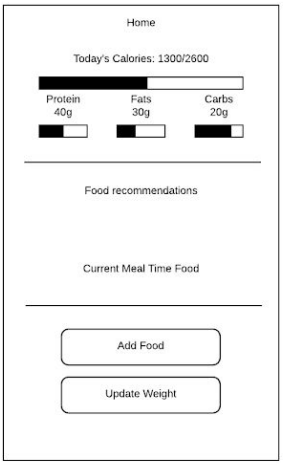



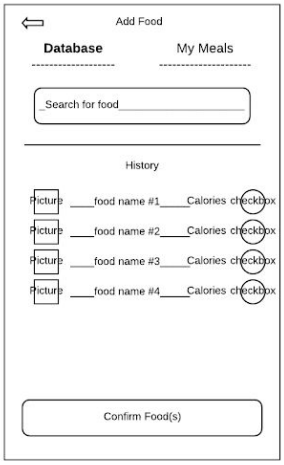

App User
: User Interface Class
: Self-Evaluate Weight Class
: Total Weights Class
: Daily Weight Report Class
1. Input self-report of weight
2. Transfer weight input
3. Request weight update
4. Accept weight update
5. Create weight report
6. Send weight report
7. Transfer weight report
8. Display weight report

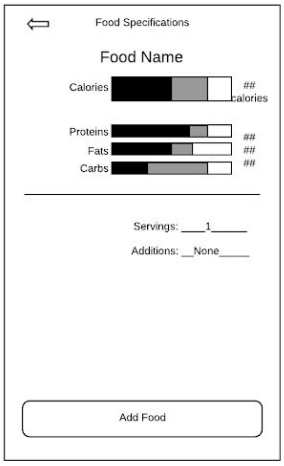

App Client
: Daily Nutrition Report Class
: Send Notifications Class
: Notification Class
1. Request daily nutrition report
2. Transfer daily nutrition report
3. Request to create notification
4. Return notification
5. Transfer notification
6. Display notification to user

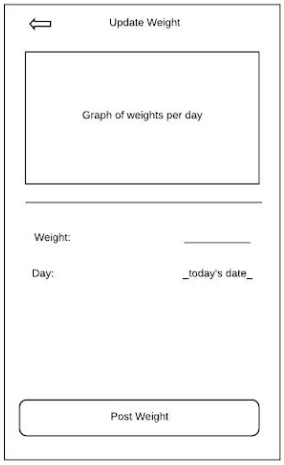
# User Interface Mock-ups

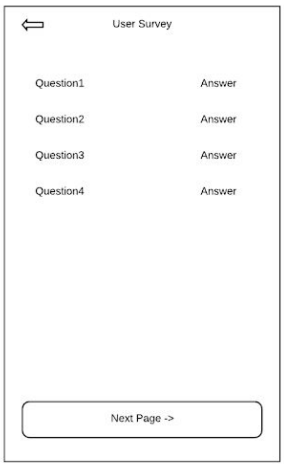




The Figure above showcases a majority of potential UI mock-ups for the Nutrify mobile application. These mockups demonstrate key design principles such as visibility. All buttons are large, non-wordy, and within thumbs reach. Any button press will immediately bring the user to a new screen unless any requested information is incorrect. A search bar is provided for querying a food database which updates on-the-fly using autocomplete. Posting a user's daily weight will update a weights graph with a new data point.

The other key design principle shown is affordance. Homepage control suggests users to manage/fill up calorie and macro progress bars. Food selection from the database allows multiple additions by filling up checkboxes and changing the text box in the "add" button from "Add Food" to "Add Foods." Updating the weights graph suggests that you should strive to follow an upward or downward trend depending on weight goal.

The UI mock-ups also attempt to align with the "Eight Golden Rules" of interface design. By striving for consistency, and while not currently demonstrated within the mock-ups, a similar palette of cooler tones + whites will make up the app (and a night mode if there is time). Buttons will exhibit the same/similar shape and placement, and font will be consistent throughout.

By enabling frequent users to use shortcuts, users adding food through the database can multi-add foods from their food history for quick additions. As a stretch goal, the client will learn the user's diet over time and recommend foods to add to finish out calorie/macro goals.

Nutrify will also offer informative feedback by having calorie and macro goal progress illustrated by progress bars on the home page. Calorie and macro intake of foods is displayed within food specifications if so desired, and will illustrate how the nutritional value of that food will affect your daily goal, along with servings and additions.

Provided in the design are dialogs to yield closure. Completing the identification survey will present the user with a completion screen showcasing their initial calorie and macro goal, and by what date they can optimally achieve their goal weight before dropping them into the home page of the app.

Simple error handling is provided for user inputs. This includes handling for user e-mail and password for log-in and registration, questionnaire answers, food database searching, weight updates, and food specifications.

Nutrify also permits easy reversal of actions, first and foremost by providing a back arrow to prevent getting stuck on one screen. If necessary, the user can remove foods added to daily nutrition if needed, as well as remove/change daily weight if needed.

To support internal locus of control, allow the user to create their own custom meals/foods that are not a part of the foods database. As a stretch goal, the app will start recommending food items to the user as it gets to know the user's diet. Give the user the option to pick and choose which recommendations are wanted by the user.

To reduce short-term memory load, provide a history of past foods eaten, past daily nutrition dates, and past weight updates. Congratulate the user on achieving their weight goal with a notification.

## Requirements Achieved

Many aspects of the project were given significant consideration while designing each UML diagram and Use Case Description. Modularity and encapsulation are particularly important in facilitating information hiding and reuse. Encapsulation is particularly important to information hiding and reuse such that many parts of each algorithm rely on many different variables, functions, and procedures segmented off into distinct encapsulated classes. Many classes exhibit inheritance, with a base class (i.e. Food ) being inherited by a subclass (i.e. Food Specification, Food Display), which results in a form of information hiding, protecting aspects of the program from others if decision decisions are changed. Many of these classes are reusable in the sense that the user will be utilizing functions that require many of a class or entering information into a class more than once. Total Nutrition and Total Weights are classes that will be accessed multiple times to store Daily Nutrition and Daily Weights classes, that will namely be created on a daily basis.

By exhibiting high cohesion and low coupling, modularity is optimized by more easily allowing the swapping the implementation of a part for another. There are less “wires” to unplug and replug when exchanging out modules. In the context of the state diagram, states like “update weight” can easily be replaced or even removed without the entire app functionality faulting.

Elegance and efficiency of algorithms are also considered. The algorithms created are made such that processing a desired action on Nutrify only takes a few clicks. The application is made with user retention strongly kept in mind, as self-help apps are strictly based on self-motivation. As with any mobile app, the user experience should be kept as easy and efficient as possible.

Again, high cohesion and low coupling is exhibited most through the state diagram, showing only the transfer of a few pieces of information between each state. States are also made to maximally hold as much functionality as their purpose can, such as the “identification survey” and “add food to daily diet” states, allowing the user to perform many actions on a single screen without the actions becoming unrelated.

Algorithms for use cases such as meal reminder notifications only look at as much data as necessary to determine whether a notification should be sent. The app client approaches the user’s total nutrition and pings for whether an update has been made in the last range of meal time. This one piece of data is all that is needed to determine the need for a phone notification, and will be sent to the user accordingly.

The data structures utilized for Nutrify were chosen with purpose in mind. The different data structures used within Nutrify can be seen in both the state diagram and design class diagram.

Within the state diagram shows the need for a food storage database for the user to query from in order to select and add the different foods they eat for meals throughout the day. This will be implemented as a PosgreSQL database for the user to query from, and will contain a multitude of "Food" objects coupled with "FoodSpecifications" that will hold the caloric and macronutrient values needed to evaluate the user's daily and overall nutritional health.

On the topic of other notable data structures, the total recorded nutrition and total recorded weights of the user will be held within a hash map, with keys represented by the dates of recording. Each map will hold a daily nutritional and daily weight object holding the information recorded by the user for that day; the user will be able to access their nutritional history if they so desire. A hash map will be especially useful for graphing weight update data on the weight updates page, as each key, being the date, can be represented on the x-axis, and updated weight values can be visualized on the y-axis.