**Fall2016Swe573\_DietApp\_Michael\_Boyman**

SWE 573 Fall 2016 Class Project.

In this document, I will explain the development process and installation of the web application and provide additional documents prepared for the project in conjunction with the project plan and revised requirements.

**Project Architecture**

A short summary of the Project Architecture can be found below:

* Server side and overall backend development has been made in Python as a deployable .py application, to be deployed on initially Django Framework, then bottle.py Web Container.
* Client-side is a fully functional html page, client side scripting is completed using J*avascript*, Angular JS and injection inside the HTLM is made using gulp.
* To hold the information inside the database, I have used simle JSON format that can directly be implemented and read by *MongoDB*.
* Issue tracking has been implemented dually by GitHub and Trello.

**Project Wiki**

Starting from the beginning, two wiki pages have been created and kept up to date. The first one defined the Project Elicitation and includes all the revisions and implemented features for the project. The second one is a weekly progress report of what has been done, and describes how the project has been implemented on a more detailed fashion.

# 1.Requirements Elicitation

## System definition

The project will consist of a web application to track food consumption and activities.

A prospective user should be able to register and create an account. The account should be created by asking the following questions:

* Name
* Surname
* Mail address
* Gender
* Birthdate (from a date picker)
* Height (in cm / in in)
* Weight (in kg / in lbs)

The system will have an authentication / authorization system to login to the dashboard. Once a user logs in to the system, he/she should be able to update his/her status for every meal. On each update user should update:

* Food type consumed (from a list, list DB: <http://www.nutristrategy.com/activitylist4.htm>) (mendatory field)
* Food quantity consumed (mendatory field)
* Weight (mendatory field)
* Notes (optional field)

update his/her status for every activity. On each update user should update:

* Activity type performed (from a list, list DB: <http://www.nutristrategy.com/activitylist4.htm>) (mendatory field)
* Activity time / unit (mendatory field)
* Weight (mendatory field)
* Notes (optional field)

After each user input, the system will calculate the user’s current Body Mass Index such as:

BMI = (Mass (kg))/〖Height (m)〗^2

Each meal should be logged as a list, with a timestamp as a header and logged per day. (@ Access through header, #Title: My Foods)

Each unit of activity should be logged as a list, with a timestamp as a header and logged per day. (@ Access through header, #Title: My Exercises)

The system should have a reporting tool to show nutritional trends from date A (from date picker) to date B (from date picker) (@ Access through header, #Title: My State)

* One line on the graph (y-axis) should show calorie intake
* One line on the graph (y-axis) should show calorie expanse
* X-axis should show time, with a daily precision

# 2.Weekly Progress Report

## Tuesday, October 4th

Initial commit is made,

* Includes a software elicitation document that overlooks the project,
* Includes a UI mock-up, to outline the general state of the pages.

### Tasks Completed:

1. GitHub Account Credentials Renewed and Desktop Application Integrated to all Workspaces
2. Python 2.7 Installed, with Tornado Framework, however it was less popular among users and test codes were not as fast as expected.
3. After online research, Python 3.4 Installed, with Django Framework, coupled with SqLite, and tasks performed was completed faster then expected.

## Tuesday, October 11th

Access to source code was reestablished through Cutu ([www.cutu.io](http://www.cutu.io/)).

* System infrastructure was evaluated with UI requisites.
* Sample websites with similar authentication and daily information input was researched.

### Tasks Completed:

1. An API requirement was found and researched online. Two frameworks were found: apiary and apigee (both links are available under trello project management tool)
2. Another requirement was that it seems necessary to implement a logging infrastructure to keep track of who did what with the affiliated timestamps.
3. After online research, this information can also be stored under SqLite, MongoDB or AzureStorageTables, depending on where the platform will run eventually.
4. The website in discussion should also have an API to both intersect and interface with the data sources at usda and nutristrategy (both links are available under trello project management tool as well)

## Tuesday, October 18th

Started working on the paperwork of the project. The documents in progress for the project are:

* Class Diagram,
* Sequence Diagram,
* Activity Diagram and,
* Specific Use Case for the design in question.

### Tasks Completed:

1. Started on writing a perpetual logging infrastructure for the website and connect it to the database with timestamp and GUID's for each particular user. This system will have a CRU (Create / Read / Update) management.
2. To store the information of timestamp, EPOCH format will be used in storage over at the DB to prevent time differences in different countries.
3. After online research, I concluded that using python with Django for the backhand will work best with a front-end of JavaScript, particularly AngularJS 2.0

## Tuesday, October 25th

Finished working on the following diagrams for the paperwork of the project. (Currently published under trello)

* Sequence Diagram,
* Activity Diagram and,
* Specific Use Case for the design in question.

The following diagram took a longer time than expected to complete, so it got postponed to the next week's task list.

* Class Diagram,

### Tasks Completed:

1. Still working on writing the code for a perpetual logging infrastructure for the website and connect it to the database with timestamp and GUID's for each particular user. This system will have a CRU (Create / Read / Update) management.
2. To store the information of timestamp, EPOCH format will be used in storage over at the DB to prevent time differences in different countries.
3. Once the logging infrastructure is complete, the work on the front-end will start.

## Tuesday, November 1st

Finished working on the logging infrastructure for the website with the required technical specs. Started working on the API infrastructure for the system.

### Tasks Completed:

1. Logging Infrastructure is designed and implemented.
2. Started on working on the API Infrastructure. The most reliable source found to be <http://www.django-rest-framework.org/>

## Tuesday, November 8th

Finished working on the logging infrastructure for the website with the required technical specs. Started working on the API infrastructure for the system.

### Tasks Completed:

1. Virtual Environment was researched, and set-up (which was a necessity for the API Rest Design)
2. Python Django Rest API design has started.
3. Serializers are researched and implemented to convert/communicate JSON format data. Basically it is used as an object relational mapper (like Entity Framework).
4. Database design with the actual system inputs has been started.
5. Successfully implemented get methods using RESTful communication using Django API Infrastructure.

## Tuesday, November 15th

Django Rest API and USDA database had compatibility issues and required additional research for integration.

## Tuesday, November 22th

Issue with the USDA was found. Apparently, while requesting data for items, I was unable to insert database the nutrient details for some but not others. The problem was found to be about the JSON response from USDA API.

## Tuesday, November 29th

Continued working on the API infrastructure. Removed unnecessary details from the USDA API requests since they could be categorized under too much information.

### Tasks Completed:

The logic for API has been refined to only display the following:

1. Food information API requests are now only limited to calories acquired when a specific food is eaten.
2. Exercises information API requests return all the information, since it already only has a calorie expanse value.

## Tuesday, December 6th

Exercises API Access has been established, implemented and tested.

* Strategic decision was made as such: All information from <http://www.nutristrategy.com/activitylist4.htm> has been downloaded and saved to the project database in JSON format.
* The exercises db has an array format for easy access.

### Tasks Completed:

The logic for the system has been established as following:

1. The user's weight information will be acquired and used individually for each user's specific BMI calculations.
2. The current exercises db has discrete values for specific weight values however it is very unlikely that people would have only such values, and an extrapolation had to be made.
3. The db has values with 25 pounds (approx. 10kg) intervals. So, each individual user weight is used to the closest value they have in the database, and their caloriesBurned value is set as such.

## Tuesday, December 13th

Front End part has been started and implemented

* Use of Angular JS, coupled with Gulp Injection inside the HTML was made Acquired information from the Back End has been connected with the rest API to the Front End, and has been committed to the repository

### Tasks Completed:

1. Login and Register screens has been implemented.
2. Food and Exercises screens has been implemented.
3. BMI calculation chart has been implemented to the dashboard.
4. Requirements.txt package manager has been implemented to manage dependencies in a new environment for installations.

## Sunday, December 18th

After the presentation, feedback received that the UDSA API request should actually include all the information such as sodium content and whatnot, so the call was reverted back to the original, and now displays all the information.

### Tasks Completed:

Front-End

1. Login, Logout and Register screens has been revised, Logout now flushes the cookies appropriately and redirects the user back to the login page.
2. Food and Exercises screens has been revised, Food screen now has been divided into Food Selection and My Foods screens, where Food Selection allows the user to search for any food with keyword input, and My Foods displays all the Foods selected by the user.
3. Exercises screen has been revised, it now has a dropdown menu where the user can select and/or search by typing any exercise, and it will be added to the user's exercises performed once selected.

Back-End

1. The system has been revised to be published on the cloud, it is currently operational on the address <http://52.166.130.21:8080/static/index.html#login>
2. The platform now includes bower
3. The platform now includes gulp inject
4. The platform now runs Requirements.txt package manager

**Issue Tracking**

Issue tracking will be handled through issues opened on GitHub and once resolved will be closed and archived under the same repository. Daily chores and issues that arose from simpler tasks will be determined and handled under Trello, and catalogued under Completed when finished.

**Contributions**

Since this is a single-person project, the development, deployment and revisions will all be completed by the same person, therefore all issues, bugs, requirement-engineering will be handled by the repository owner. Therefore, there will not be division of labor.

**Deployable Final Product**

Deployable final product can be found online under the address <http://52.166.130.21:8080/static/index.html#login>

**How to Run this project**

Since this is a completely open-source project, anyone should be able to fork, develop their own version and run this project.

The system is designed to be very simple to install and run from the beginning to a freshly installed environment.

To start with, you would need to install python3 into your system. In linux, and Mac environments, it comes preinstalled. Final version of Python can be acquired from the following link:

https://www.python.org/downloads/

Once we have installed python, we will need the package manager tools (pip), which can also be installed with the following command tools:

On Linux or OS X:

pip install -U pip setuptools

On Windows:

python -m pip install -U pip setuptools

Further detail can also be found under the following link:

https://packaging.python.org/installing/

Once we setup the minimal infrastructure, the system requirements can automatically be installed and implemented by inputting the following command:

pip3 install -r requirements.txt

Afterwards the system will automatically download and install all dependencies and will be ready to deploy.

Finally we will run the main application as:

python3 main.py

And we will be operational.