# Healthify

# Just Deserts

Just Desserts

Joey Hernandez, Thomas Kennedy, Riley Mapel, Alek Sevilla

Capstone Project Fall, 2023

The University of West Florida

Dec 1, 2023

CIS4595 - Capstone Project

Bernd Owsnicki-Klewe

## Abstract

Within this deliverable is our documentation of our capstone project “Healthify”. We have a product description that is a high level description of what our application does and how a user could use it. We have a comparison of our initial timeline and final timeline of how our project schedule went. After the timeline we have our use cases we created to help with development with a user in mind. After our timeline we have our software evaluation in which we have our software functionality and security overview. Following the software evaluation we have the section of work that is to be completed or additions we would have liked to add onto the project if we had extra time.

# Table of Contents

## List of Tables

## A. Project Description

Our project is a web based shopping service with incentives built in to advocate for healthier options when it comes to purchasing food items. In order to accomplish this we will have built-in rewards that will provide the incentive to select healthier food options with an added benefit of earning more rewards for better/healthier food selections. Items within the application will have a grade given to them and if a user selects an item with a higher health rating they will also receive more reward points that can be used for a later order

## B. Final Timeline and Comparison With Initial Timeline

*Table 1* : Timeline Comparison

| Sprint | Dates | Original Plan | Finalized Plan |
|---|---|---|---|
| 1 | 9/11 - 9/25 | Skeleton code for look and feel of site. | Skeleton code for application and API implementation (script). |
| 2 | 9/25 - 10/9 | Logic built in to develop functionality for site. | Adding functionality to our application (search, sign in, cart, etc.) |
| 3 | 10/9 - 10/23 | Reward system developed for users. | Tune up functionality from previous sprint. |
| 4 | 10/23 - 11/13 | Profile development for individual users within application. | Database integration for storing and synchronizing user information (cart, bal, points). |
| 5 | 11/13 -11/27 | Secure logins and logouts along with quality-of-life changes. | Checkout development, Implementation of rewards to products, styling. |

## C. Project Results Compared With Expectations

**Use case 1**: User browsing and selecting items from Healthify

**Actor**: Regular user.

**Goal**: Browse items, add items to cart, and purchase items.

**Steps**:

a. The user opens the Healthify application.

b. They navigate through product categories using the search bar and find a product they are interested in.

c. Upon clicking on a product, they are redirected to the product details page, where they can view more information about the item.

d. The user decides to add the product to their shopping cart.

e. They continue browsing and add a few more products to the cart.

f. The user clicks on the shopping cart icon to review their selected items.

g. In the shopping cart, they can see a list of selected products, their quantities, and a total price.

h. The user clicks the "Proceed to Checkout" button.

i. Review the order, and confirm the purchase.

j. The application processes the order, deducts the items from inventory, and provides an order confirmation to the user.

**Outcome**: The user successfully browses, selects products, and completes a purchase on the Healthify platform.

**Use case 2**: User uses account for rewards

**Actor**: Regular user

**Goal**: Login using their Google credentials and have reward points linked to their account.

**Steps**:

a. The user opens the Healthify site.

b. The user clicks the sign-in button.

c. The application opens a Google authentication window.

d. The user provides their Google credentials; and upon successful authentication, the Google authentication window closes.

e. Once logged in, the user starts browsing products and adds items to their shopping cart.

f. Healthify has a rewards program that assigns points based on the user's purchasing behavior.

g. As the user continues to make purchases, they accumulate rewards points.

h. The user can view their rewards balance and potential discounts within their profile or during the checkout process.

**Outcome**: The user successfully logins with a google account, and can earn rewards with their purchases.

## D. Software evaluation

### Functionality

While the testing plan ultimately devolved into manual code testing during production, the site still came together very well and functions as expected with no outstanding issues that hinder usability and accessibility.

The plan for testing and evaluating the functionality of the site originally included the implementation of a testing suite called Vitest. That test suite would be used to write a series of tests for planned code. Of the planned tests, only one was ever actually written. Fortunately, the manner in which this project was being created ultimately made the planned tests unnecessary as the code was being tested by the writer of said section of code. All code was tested manually by the writer for functionality before being committed to the rest of the code. Given the manual testing of code being conducted all the time, everyone was getting a chance to ensure code functionality.

As the testing lead, a lot of my contributions were in the form of quality control and user experience. The core functionality of the code was made less of a concern due to the aforementioned circumstances that led to functionality being tested at all levels of implementation. Taking the time to simply use the site as a user revealed a handful of odd design decisions and missing functionality from certain pages that were rectified upon discovery.

As of the writing of this report, there are no outstanding functionality issues. The codebase is very robust and holds up to both normal and strange user behavior well. While having the appearance of a student project, the site still flows and functions very well and behaves as one would expect.

## Security

- We used the security feature with Google Cloud Console, to find some vulnerabilities, as well as an application that checks for vulnerabilities on our web application, it came back with no major vulnerabilities.
- We found vulnerabilities like:
  - An ssh server that could be accessed by anyone to a database. - fixed
  - User from outside of our domain (uwf.edu) - didn't need to be fixed
- The database where login info is stored is encrypted by default to prevent unauthorized people from accessing it.
- A security issue we still have is that all of our API keys are public, this can't be helped because we don't have a separate server to keep things hidden.
- The user's name assigned to the gmail account is stored in the database, meaning that if there were to be a breach, a hacker will get the names of the users.
- We don't have any security headers, this can make us vulnerable

Other Security Documents:

Web Application Security 10/26/2023

Broken Access Control - Right now anyone can access the site because we have a site hosted by github. Our repository is public and it can be accessed at https://justdessertscapstone.github.io/shop/ right now we have no difference for users, and no accounts are made at the moment. All users will see their own version of the same page, and each user will have their own things like cart, searches, and purchases, this will be configured through the client ID that will be created when a user logs in with google authenticator.

Cryptographic Failures - Default creation of the Client ID is the user's name and then 6 random numbers, we will add a hash onto it to make it much more secure, this will help ensure the user's data at rest will stay secured. When we get to the part where data is in transit, we will ot transfer it in plaintext. Also, I think it would be best to not have an option to store payment information, we should discard it as soon as we are done with it.

Injection - First things first, all user inputted data will be validated, filtered, or sanitized to protect against injection attacks.

Insecure Design - We are using a secure design, we will do plausibility testing, and we have set limits on the oauth and database.

Security Misconfiguration - We have nothing unnecessary in our project, everything is minimal, and all things that we have rendered obsolete, we have deleted (functions that were not efficient).

Vulnerable and Outdated Components - We are only using up to date components.

Identification and Authentication Failures - We will not be using anything like 2fa for our application, but we are using google oauth, there will be no usernames or passwords, it will be through google.

Software and Data Integrity Failures - We are only using trusted libraries for npm, also we will look to add digital signatures.

Security Logging and Monitoring Failures - We will be logging and monitoring our system.

Server-Side Request Forgery - We will be denying by default, all data inputted will be validated and sanitized.

Healthify Security Documentation

Security Requirements:

Authentication and Authorization - We use Google OAuth for signing into our application and users that join do not have any permissions that would cause vulnerabilities.

Data Encryption - The login information is encrypted when in transit and in rest, this is done by default by Google OAuth and Firebase (our database).

Secure APIs - All APIs that we use are secure and trusted.

Regular Security Audits and Testing - I ran web vulnerability scanners on our web application, and there were no major vulnerabilities.

Data Protection Methods (Encryption, Hashing) Details:

Encryption - The only data we have that needs to be encrypted is the login, that is all handled by Google OAuth and Firebase. Both of these tools that we use for moving and storing data encrypt the data with an AES-256 encryption algorithm, it would take a supercomputer millions of years to brute force crack it. Firebase also uses the AES-256 encryption algorithm for the data at rest, making it a secure way to store all of the data.

Secure Coding Standards:

Safe Failing - We use error handling to make sure when there is an error, the user will fail securely.

Minimal Privileges - All users have the bare minimum privileges to be able to function.

Code Verification:

Code Review - Each member of the group verifies that the code meets security criteria.

Dynamic Analysis - We run the application and test various conditions to observe the behavior of the application, and if it is not how we expect it to be, we change it.

Security Scanning - We scanned our application through a web vulnerability scanner and it came back with no major vulnerabilities, the only thing was a lack of security headers.

Security Testing:

Security Scanning - We scanned our application through a web vulnerability scanner and it came back with no major vulnerabilities, the only thing was a lack of security headers.

API Security Testing - The user has no ability to access any APIs, this is done so that there is no threat for a bad actor to attack the API with an inject attack.

Summary:

Healthify is a secure web application, we know this by running our application through a web vulnerability scanner and finding no major vulnerabilities. We also have encrypted all data that is in transit or at a rest state, the encryption method is AES-256 which is a very complex and secure encryption algorithm. All users have minimal privileges, and we made sure that users will fail securely if they fail. We also used secure coding practices and all of us checked each other to make sure the code we write is secure.

## E. Work to be done

- Make ‘Navbar’ for suggested searches.
- Make the site look more professional/polished.
- Finish test cases.