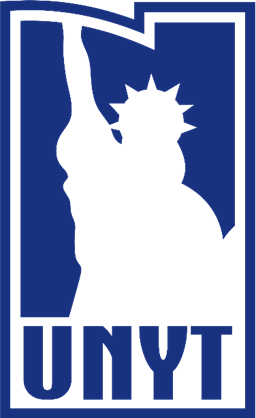
**Software Engineering**



**University of New York Tirana**

**Project**

**Kristo Cavo**

**Food Delivery Website**

**Table of contents**

1. Analysis and Design……………………….……….………………………………………
2. Implementation…….…………………………………….…………………………………
3. Version Control…….………………….……………………………………………………
4. Testing…….………………………………………………………………………………...
5. Project Management…………………………………..……………………………………
6. Conclusion…………………………………….…………………………………………….

# Analysis and Design

This website is designed to give the customer the option of food delivery straight to their home. To be able to achieve this, several requirements need to be met. The scope of the project is humble, as the only real functionality that will be available is the purchase of food articles from the client and the maintenance of the website from the administrator.

**General Description:**

General functions:

* The user of the website will be able to pick any item from a menu. Once the order is filled out, they are also able to give any specific instructions regarding the order.
* The administrator of the website is able to edit the menu according to what is physically available at the moment.

**Feasibility:**

This type of project has been previously proven to be a success. The reasoning behind it is that the success of the business itself is also due to the quality of service. In this case the website is used as a marketing tool. A good, easy to use website can have a great impact when it comes to sales and marketing.

From the perspective of a software engineer, this website can be planned out relatively early, as the requirements are not extremely complicated for a company of this scale.

**Target audience:**

There is no specific target audience for this food delivery website. That being said, design-wise, the website is more catered to the younger generations with a simplistic, clean look.

**Nonfunctional requirements**

### **Usability**

This web application is very easy to use, it will be very easy for user to learn and operate the system. Also, the admin interface is built in a way to be easy to use.

**Efficiency of use:**

**The average time it takes to accomplish the user goal (ordering food) it minimalist, so the user can complete its goal without any help, and the number of transactions completed without errors.**

**Security**

Security requirements ensure that the software is protected from unauthorized access to the system and its stored data. It considers different levels of authorization and authentication across different users’ roles. So, the example here: the admin interface can not be used by unauthorized persons.

Functional reequipments

The can user sees the entire menu, prices and description.

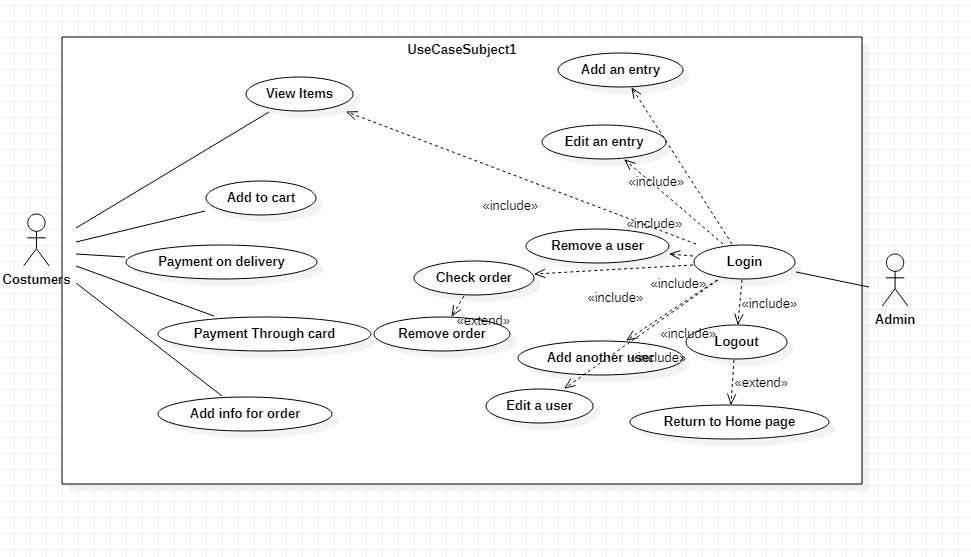
User can complete the payment in a convenient way for them (payment on delivery, payment through card)

User can specify any special requests regarding their order, for whatever reason.

Admin can log in into the website quickly and reliably.

Admin can edit the menu based on any specific business reason.

**Use case diagram**



**The website itself is going to be split into 3 main pages:**

* Home Page
* Order Page
* Admin Login Page

**Home page:**

The home page will be designed in a minimalistic manner, providing as much information as possible without cluttering the view. It will be split in two main halves, the left being the interactive half of the page, where the customer can see the different ways to complete delivery and the total of their order. The other half of the page will display the menu with prices and quantity.

The customer should be able to add items on the menu as they please, and the website will respond accordingly in real time.

**Order page:**

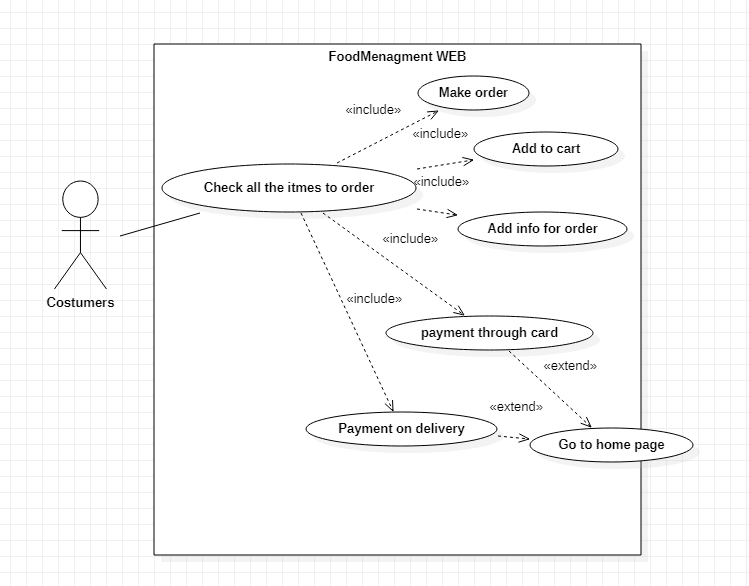
The order page’s purpose is to collect delivery information and payment information.

The order total carries over from the previous page and once the confirmation has been given, the total is saved in the database along with any other information required.

The payment is optional, as the customer should also be able to pay on delivery, but the feature is there for convenience purposes.

**The below use case diagram shows a general idea as to how the order process goes:**

**Use case diagram Users**



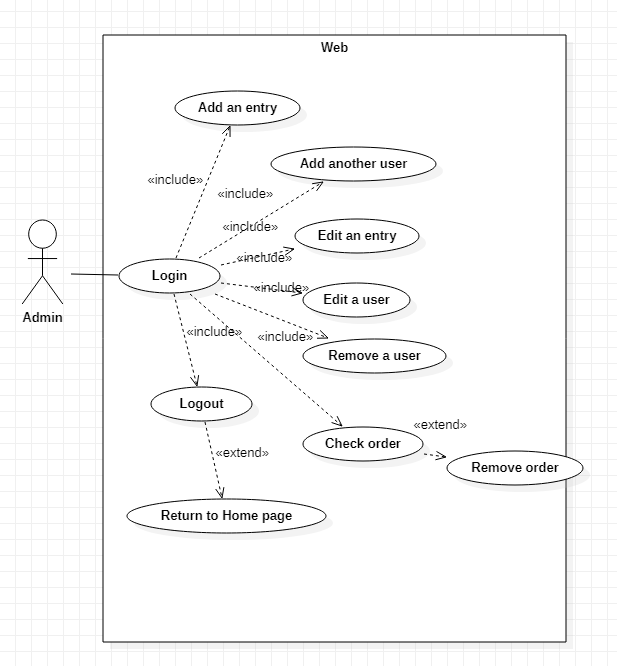
Note:

The website holds no payment information for privacy purposes.

**Admin page:**

The admin page’s main purpose is to moderate the website and update information based on business requirements. An admin has the ability to add users and menu entries and also remove them accordingly. The admin however, cannot add new orders, he can only remove invalid orders.

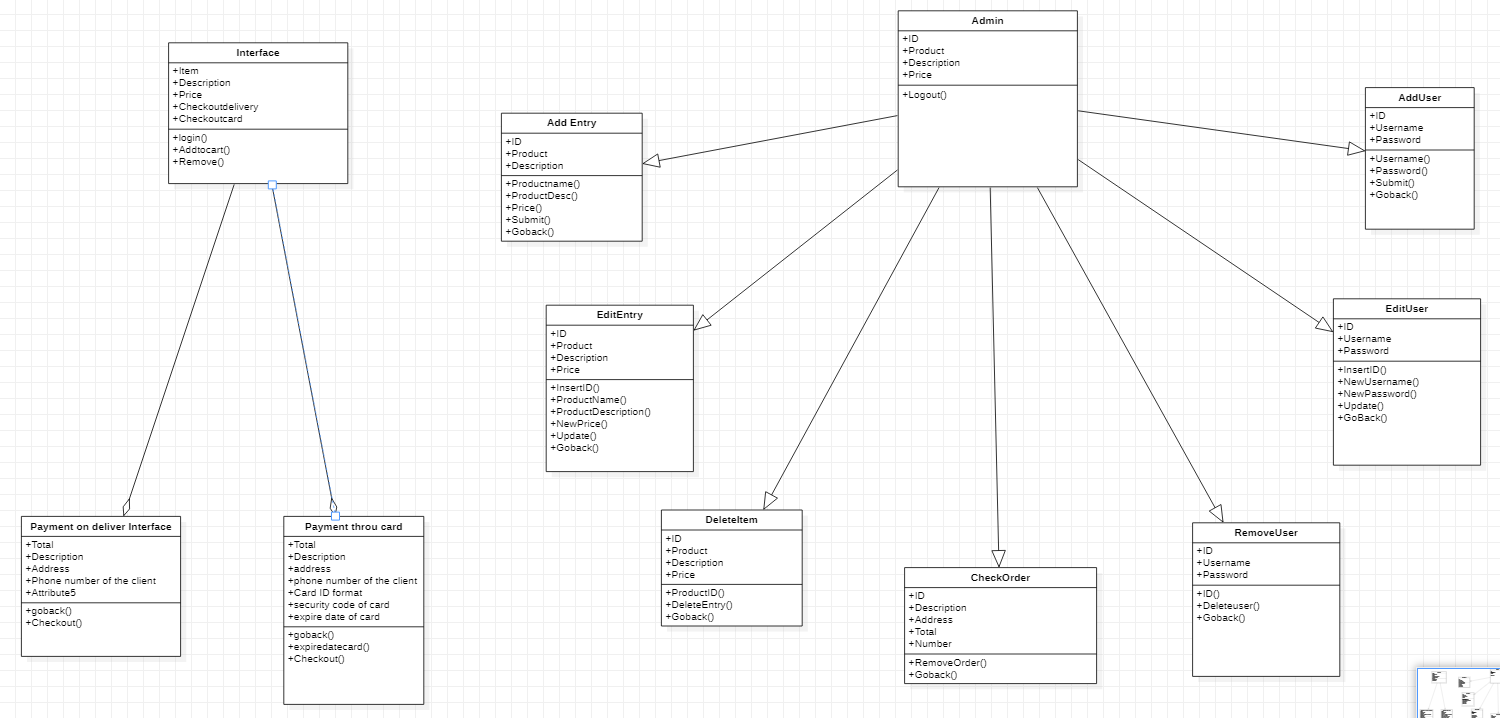
**Administrator Use case diagram:**



The below diagram shows the detailed structure of the website based on the class structure of the website.

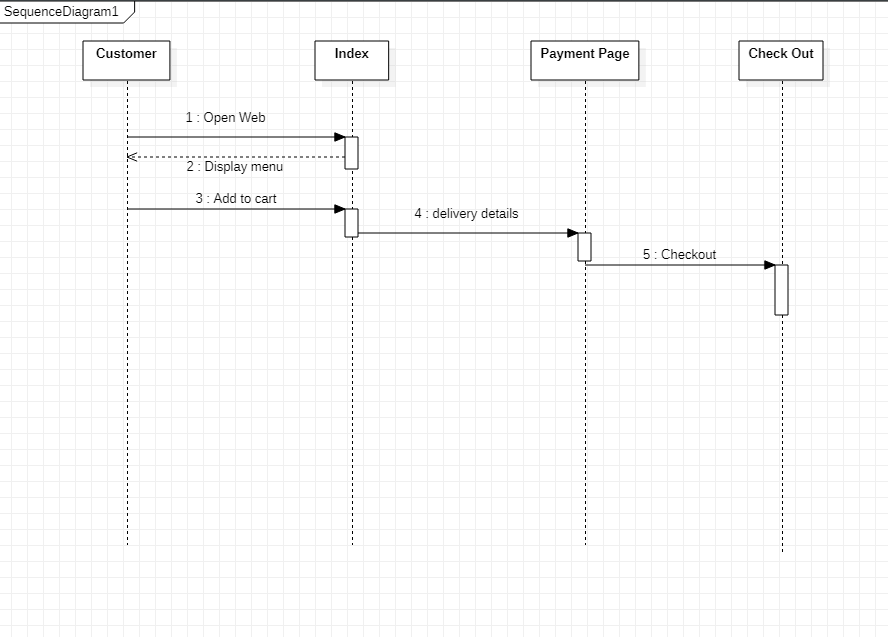
Each class is shortly described with their respective properties and classes.

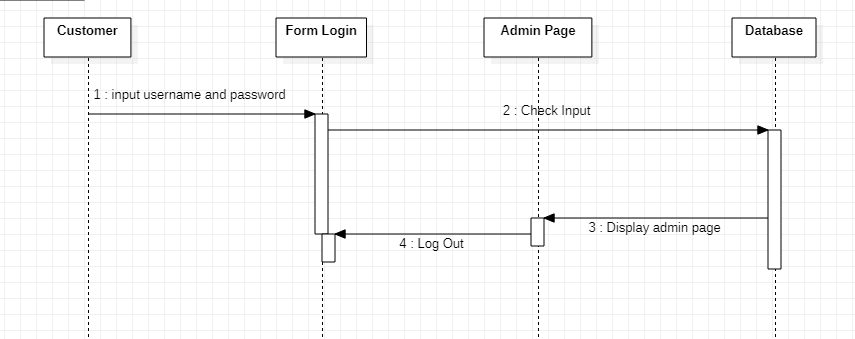
**Class Diagram**



**Sequence Diagram**

**Customer Order Path:**

****

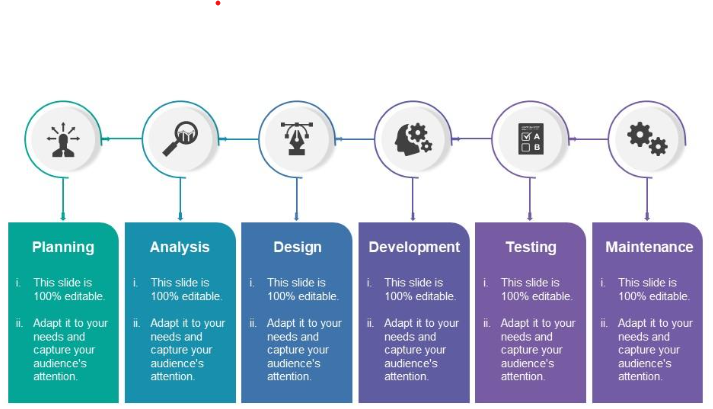
**Login sequence diagram:**

# Implementation

The website is written mainly in PHP and JavaScript. HTML is used for website structure and the styling is done through pure CSS, without any specific library.

This means that it’s relatively easy to implement the code, as no specific assemblies or references are required to get everything going.

The implementation of each feature has been done in this specific order.



Once the general system analysis and design was completed as defined above. The development was the part we were most familiar with.

The tasks were split evenly and based on what our team was most comfortable with. This was part of the planning phase. The splitting of the work is mentioned below in the last section under project management in greater detail.

Testing will also be discussed later on in its own section.

That being said, for maintenance, the general design idea is that a specific support contract would be made with clauses under how long the support will continue and what it will cover. This was, on the business side, as developers or functional analysts we would not have to deal with unrealistic and uncompletable requests. Although, that remains as a responsibility of a business analyst, instead of a software developer.

The website implementation after development and testing is very straightforward. Once the code has been downloaded/imported, all one has to do is paste the code into the specific folder that XAMPP (in our case) defines for a quick demo.

Once the code has been moved to the correct location, the website should be up and running.

**The documentation does not cover the publishing of the website online.**

# Version Control

For version control we have used GitHub. The repository has been created after full development of the application was completed, therefore there is no version history or multiple commits. While a local git was present for version control purposes and efficiency of not losing code, we never made the effort to fully commit it online after complete development.

The link for the repository:

https://github.com/dejvss/foodmanagement

# Testing

The testing strategy we used for this project was continuous.

By that, we mean every single particle of code we use, we tested in the same time in order to see the ability of our creation.

We split our environment into two same ones, one being the developing environment and one being the testing environment. This way, we could work without being interrupted by any specific outside factor, and whenever needed we could update the changes over on the test environment and be done. Each developer took care of a specific 5-day task. We begin with user-interface which took us around 5 days just to create and we took around to 4-5 days testing to check if is there any problem, code bug or even grammar mistake that could interrupt our process. Around that time, we manage to finish our front-end of the program and continue with the other part of it.

The developer of the specific feature/functionality would finish that specific part of the development and the other developer would then receive the changes, and test them on his own time. This enabled us to keep track of each other, and at the same time do some quality assurance when it came to our code. I would refer that front end was probably the easiest part of our program. The hardest and where we started to get dead-end was in the backend and the association with database, there we had to do a lot of stuff and a lot of research till we manage to procced with a strong and secure code that was bugless and non-false. (You could say that we did a manual test for the program in every particle and part of it in order to check the reliability of it). Afterwards this part we start to fix a little bit our web and project architecture because we found a lot of confusing things and a lot of discrete codes. This project is our best solution to a simpler and clearer project.

We also wrote down multiple comments for the other developer, so as not to be confused if there was any misunderstanding. This way, bugs could be solved by either developer if good communication was present. This would in the future help us to get more comfortable when editing something specific, as we would not hesitate to edit the code.

Unit tests were used for the more important functionalities of the website, such as the checkout functions. We needed to ensure that the functions work properly and return the correct information, so unit testing was a good way to make sure that everything was functional. That being said other functions were also tested in the same way. Specifically, admin use case functions for adding and removing things in the database.

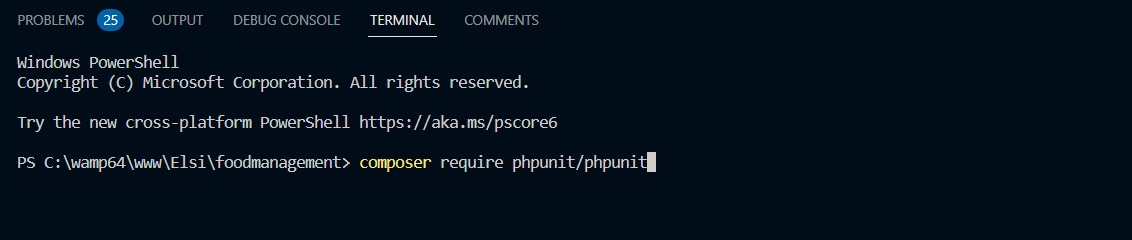
Ultimately, we found unit testing to be useful, yet slightly redundant. The scale of the project was not big enough to justify a large amount of unit tests. Functions were overall pretty easy to cover.

That being said, our project design for testing turned out to be relatively successful, as not much had to be changed other than the initial interfaces that we concepted for our elements. The original design was a bit more simplistic, but we though it to be too basic, so we agreed upon a mutual level of abstraction.

The original design included more front-end engineering though, as everything was supposed to be on one page, so it would be a one-stop-shop experience. We decided that overall, it was a much sounder decision to split the website into the menu/index page and the order/checkout page.

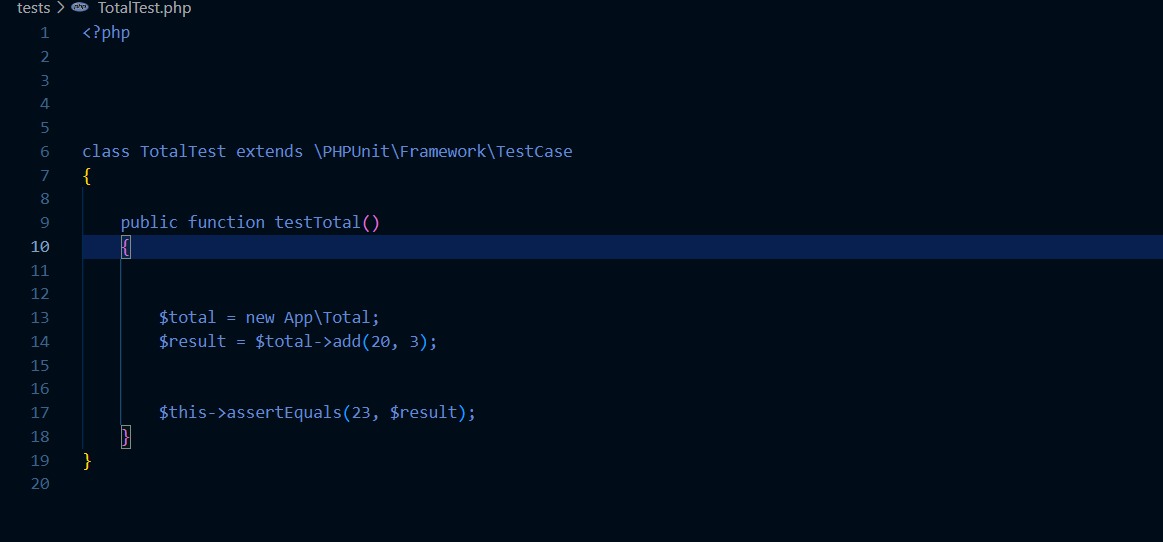
Other than that, other testing changes made were small code changes and fixes which hadn’t been correctly implemented to begin with or could use some tweaking, as the requirements were clear enough.

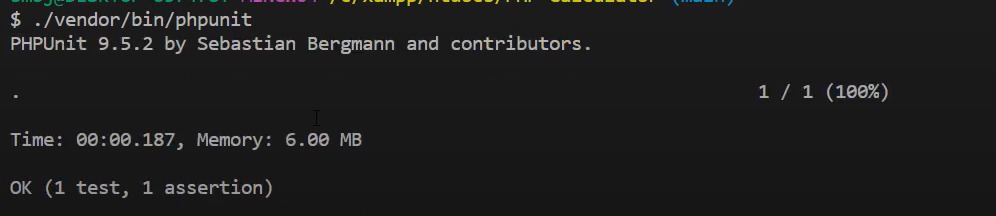
**Unit testing**

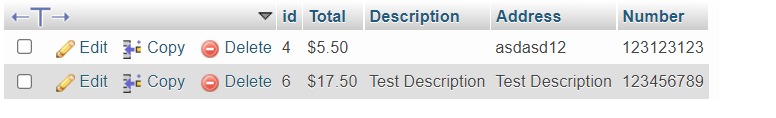
As we said we did a unit test, and we managed to get some of our methods to be a total success, no error at all.

First, we installed composer phpunit in terminal

As we notice the library is installed and now, we write the classes to test the total function, that is the main function to show the total of order for a client.

****

****

****

The unit test was a total success

# Project Management

To be able to efficiently split the work between us we have split the work based on the requirements below:

As a team of two people, we have evenly split the work between development, testing and documentation activities. Below there is a general list that indicates the flow of work.

System analysis was the first task that was handled. Since we had some prior experience, we were able to handle the task on time. We can confidently say that the time we spent on system analysis turned out to be a lifesaver in the future, as anytime we had things that were unclear, we could always rely on the diagrams to remind us what to complete next with a rough idea as to how it works.

Since we had a lot of time for project development, we have split most tasks into 5-day deadline intervals. We found it to be an appropriate time to complete tasks.

Development was left to be a 10-day work time, as time allocation for research was also needed. We had the advantage of being in a two-person group, which can be more easily managed than a higher number. This enabled us to split work evenly and efficiently.

A great deal of time was also spent testing the software as we opted for a continuous rigorous testing of features, where the other project member tested the other’s features after completion. This helped us get things done quickly and with more creativity, as having another developer test your code can really bring out any problems.

# Conclusion

Overall, we found this to be a very interesting experience when it comes to the design of software. It was a simulation of how a software engineer operates in day-to-day activities and there was a lot of insight to be gained when it comes to project management. We found the model used to be a success, as the time allocated and given each task gave us enough incentive to work at a rapid, yet comfortable rate. The project design turned out to be easy to work with and understandable.