

College of Sciences & Liberal Arts  
Department of Computer Science

**CS 471 Software Engineering**

**Winter 2022**

**Course Project**

**Final Document**

**For**

**Restaurant Ordering System**

**Version 1.0**

3/18/2022

**Table of Contents**

[**P**roject Proposal](#_caydjaknahix) 3

[Software Project Management Plans](#_9bfxw9sbsr9s) 6

[Software Requirements Specification](#_yu3y45xuvad2) 29

[Software Design Specifications](#_72kqkn1svzoo) 41

[Software Test Plan](#_ozjy3eb5db01) 68

[Lessons and Skills Learned](#_wdup8rptorlc) 86

[Conclusion](#_na9ybct1kzxu) 88

[Appendix](#_uf5fsstws6ou) 89



College of Sciences & Liberal Arts  
Department of Computer Science

**CS 471 Software Engineering**

**Winter 2022**

**Course Project**

# Project Proposal

**For**

**Restaurant Ordering System**

**Version 1.0**

1/18/2022

* Project Group # 3
* Members of group

| Name | Email |
| --- | --- |
| Owen Bickel | bick7185@kettering.edu |
| Thomas Hayward | hayw6218@kettering.edu |
| Ethan Knott | knot1657@kettering.edu |
| Cameron Manning | mann6441@kettering.edu |
| David Xia | xia0627@kettering.edu |

* Title of project:

Restaurant Ordering System

* Introduction of the Project - Problem Statement

According to touchbistro, an online restaurant management vendor, using an outdated ordering system can harm a business in many ways. For example, a modern ordering system allows you to easily and correctly take orders. A modern ordering system also allows for staff to check out customers and pull up orders without having to check handwritten notes or make constant trips to the kitchen. Allowing staff to quickly and efficiently take orders without the fear of losing or messing up an order will increase productivity and possibly reduce staffing costs.

* Objectives (Product Characteristics and Requirements)
* Allow the user to easily and effectively take customer orders.
* Have multiple buttons that sort food types (EG: beef, chicken, and pasta).
* Allow for extras or a la carte items to be added to an order
* Allow for multiple orders to be taken within one instance of the program.
* Allow food item customization for entrees (if applicable to the food)
* Project end user

Restaurants and the staff that work the register. Restaurant staff would be the end user that would be using the system the most.

* Summary of Project Deliverables
  + The program is written in Java 17 and utilizes a GUI that allows restaurant employees to take and cashout orders.
  + The program should allow multiple orders to be taken before a customer is cashed out.
  + Allow restaurant staff to easily take orders and modify food items, for example, removing lettuce from a burger.
  + Include an a la carte section like side items and specialty drinks.
  + The program will write food and drink items to a text file for easy documentation of completed orders.
  + Standard documentation of the code to ensure that future programmers can work on the software quickly and effectively.
* Project Success Criteria
* The GUI is quick to respond and easy to use so that customers can have their orders taken quickly and reliably.
* The program encompasses a majority of possible requests that a customer may make so that there is never a need to handwrite an order.
* In the event a mistake is made it is convenient to undo that mistake or restart the order.
* The orders will be backed up to a text file to keep a convenient record of orders.
* Solution methodology
  + Waterfall Method
    - We do not have any obligation to a customer with the tendency to change requirements. Thus a linear and sequential system of development is the most simple and thus most effective solution for this project.
    - Even in the event we do need to make changes we are dealing with an extremely small project and a small team that will not have much trouble adapting to change. As manpower scales that process becomes exponentially more difficult but given our small size it isn’t nearly as much of a concern.
    - We considered doing a prototyping process model but given this is a school project with an already fairly low bar for the final product the temptation to submit a prototype would potentially be a problem for team motivation and quality of the final product as the term gets busier.

source: https://www.touchbistro.com/blog/cash-register/#subheading-2



College of Sciences & Liberal Arts  
Department of Computer Science

**CS 471 Software Engineering**

**Winter 2022**

**Course Project**

# Software Project Management Plans

For

Restaurant Ordering System

Version 2.0

CS 471 - G3

Dr. Jamal Alhiyafi



1/25/2022

This Software Project Management Plans (SPMP) was prepared and provided as a deliverable for Software Engineering, 471, Winter 2022, and it will be used by family-owned restaurants in need of a simple but effective ordering system.

This document is based in part on the IEEE Recommended Practice for SPMP Descriptions.

| Name | Email |
| --- | --- |
| Owen Bickel | bick7185@kettering.edu |
| Thomas Hayward | hayw6218@kettering.edu |
| Ethan Knott | knot1657@kettering.edu |
| Cameron Manning | mann6441@kettering.edu |
| David Xia | xia0627@kettering.edu |

**Table of Contents**

[Revision History](#_1i7al2yvmz7r) 6

[Table of Tables](#_1fob9te) 7

[Table of Figures](#_3znysh7) 8

[Project Overview](#_2et92p0) 9

[Purpose, Scope, and Objectives](#_tyjcwt) 9

[Assumptions, Constraints and Risks](#_3dy6vkm) 9

[Project Deliverables](#_1t3h5sf) 10

[Schedule and Budget Summary](#_4d34og8) 10

[Evolution of the Plan](#_2s8eyo1) 10

[References](#_17dp8vu) 11

[Definitions and Acronyms](#_3rdcrjn) 11

[Document Structure](#_26in1rg) 11

[Project Organization](#_lnxbz9) 12

[2.1 External Interfaces](#_35nkun2) 12

[2.2 Internal Structure](#_1ksv4uv) 12

[2.3 Roles and Responsibilities](#_44sinio) 13

[Managerial Process Plans](#_2jxsxqh) 15

[3.1 Start-up Plan](#_z337ya) 15

[3.1.1 Estimates](#_3j2qqm3) 15

[3.1.2 Staffing](#_1y810tw) 15

[3.1.3 Project Staff Training](#_4i7ojhp) 15

[3.2 Work Plan](#_2xcytpi) 16

[3.2.1 Work Breakdown Structure](#_1ci93xb) 16

[3.2.2 Schedule Allocation](#_3whwml4) 16

[3.2.3 Resource Allocation](#_2bn6wsx) 16

[3.2.4 Budget Allocation](#_qsh70q) 17

[3.3 Project Tracking Plan](#_3as4poj) 17

[3.3.1 Requirements Management](#_1pxezwc) 17

[3.3.2 Schedule Control](#_49x2ik5) 17

[3.3.4 Quality Control](#_2p2csry) 17

[3.3.5 Reporting](#_147n2zr) 18

[3.3.6 Project Metrics](#_3o7alnk) 18

[3.4 Risk Management Plan](#_23ckvvd) 18

[3.5 Project Closeout Plan](#_ihv636) 19

[4. Technical Process Plans](#_32hioqz) 20

[4.1 Process Model](#_1hmsyys) 20

[4.2 Methods, Tools, and Techniques](#_41mghml) 21

[4.3 Infrastructure](#_2grqrue) 22

[4.4 Product Acceptance](#_vx1227) 22

[5. Supporting Process Plans](#_3fwokq0) 23

[5.1 Documentation](#_1v1yuxt) 23

[6. Additional Plans](#_4f1mdlm) 24

## Revision History

| **Name** | **Date** | **Reason For Changes** | **Version** |
| --- | --- | --- | --- |
| Cameron Manning | 1/25/22 | Filled out reasonable chunk of the beginning and did basic formatting | 0.1 |
| Owen Bickel | 1/25/22 | Revised some project risks, made various grammatical changes | 0.1 |
| Thomas Hayward | 1/25/22 | Reviewed the document and filled out the Supporting Process Plans | 0.1 |
| Owen Bickel | 1/26/22 | Completed Technical Process Plans | 0.1 |
| Thomas Hayward  Ethan Knott | 1/26/22  1/26/22 | Completed some and reviewed the Managerial Process Plans  Filled out chapter 3 | 0.1  0.1 |
| Thomas Hayward  Owen Bickel | 3/14/22  3/18/22 | Filled out the abbreviations and added in proper references  Grammar errors, establishing formatting consistency, and revision of goals | 2.0  2.0 |

## Table of Tables

* [Table 1 - Week by week project goals](#_b6sfs6cu8uvo)
* [Table 2 - Job assignments for each group member](#_x14gxkl38008)
* [Table 3 - Work Breakdown Structure](#_a2smxyciq50n)

## Table of Figures

* [Figure 1 - Group communication with professor](#_3exz5dxjoc4u)
* [Figure 2 - Format of the waterfall project management plan](#_y5asq4iyjfzn)

## Project Overview

This section provides an overview of the purpose, scope and objectives of the project for which the Plan has been written, the project assumptions and constraints, a list of project deliverables, a summary of the project schedule and budget.

### Purpose, Scope, and Objectives

* The purpose of the project is to provide a cost-effective and convenient solution for smaller restaurants that want to set up ordering systems. This does limit the scope of the project. Our intent is not to compete with the major corporate ordering systems, we want to provide a simpler and cheaper solution to small businesses.
* We do not expect any aspect of the expected deliverables to not be delivered.
* Our main objective is to provide a cheap and easy solution for a restaurant ordering system. Some sub-objectives of that goal include
  + An easy to read and understand UI to ensure fast and accurate order taking
  + Simple features that will not require heavy maintenance and will be durable for years to come
* Required deliverables include the program, the code that drives the program and documentation for the code that drives the tool.
* The code will be written in Java 17. Code documentation will most likely be on the GitHub repository that a client would receive access to. As for the development method, we will most likely use the Waterfall Method. Its simplicity is reasonable for the complexity of our project and given we have an extremely small team we can be much more flexible if need be. This project is not in any way related to other projects that are being worked on by this group at this time.

### Assumptions, Constraints and Risks

* Our project completion methodology is reliant on the customer not making changes to the product. Given we do not have a customer in mind right now (and this is a school project) it is fair to assume that there will not be changes made to our project requirements.
* Project Risks:
  + Team member drops course: This is a fairly common occurrence in college environments. In the event this happens we will have to reconsider the distribution of work among our team members.
  + Homework overload: Given all of our team members are taking multiple classes there is a chance that we will all be overloaded with work from other classes. If this happens we will most likely be forced to rearrange our personal schedules to make time for this project.
  + Unexpected cost: Given we are all paying an exorbitant amount of money to attend college our budget for this project is $0. In the event we might need to spend money we will most likely have to either come up with a way of splitting the cost or work around the problem (which would waste extra time).

### Project Deliverables

* Project Deliverables:
  + Program
  + Code that drives the program
  + Code documentation (most likely done through GitHub)
* The program will be installable via the internet and access to the GitHub repository can be granted via the customer's GitHub account.
  + The customer will have to have Java installed on whatever register they are adding this software to.

### Schedule and Budget Summary

* Budget: $0 - Our project will have a budget of zero dollars because it is a school project

#### **Table 1 - Week by week project goals**

| Week # | **Goals** |
| --- | --- |
| Week 1 | Set up entrances to the ordering system and add classes to represent ingredients and various meals. |
| Week 2 | Finish core ordering system (removing ingredients, a la carte items, sauces etc.). |
| Week 3 | Make the GUI visually appealing | implement an order backup system. |
| Week 4 | Convenience features: meal shortcut for applicable items etc. |
| Week 5 | Testing, bug fixes and cleaning up code documentation. |
| Week 6 | Polish. |

#### 

### Evolution of the Plan

* The structure of this Project Plan is in compliance with the recommendations of IEEE Std 1058-1998.
* This plan will not receive scheduled updates as in an ideal situation the plan will not change. In reality there will probably be unscheduled updates that make changes to project requirements based on what is reasonable for a group of students to complete in a Kettering term.
* Updates to the plan will be disseminated via Google Doc and a notification will be sent to group members via Discord when they occur. There is a table at the top of the doc that will be used for the history of changes made to the doc.
* Given the small size of our group, changes to the plan will not be strictly controlled and changes will be at the discretion of group members. Ideally group members will hold one another accountable.

### References

* Alhiyafi, J., (2022). *Project Description*[PDF]. College of Sciences & Liberal Arts, Kettering University.
* Manning, C., Knott, E., Xia, D., Bickel, O. and Hayward, T., 2022. *Project Proposal*. Undergraduate. Kettering University.
* Project Requirements: Just reference it not link it APA style
  + Course Project, N/A, Winter 2022, Dr. Jamal Alhiyafi, N/A

### Definitions and Acronyms

* GUI - Graphical User Interface
* IDE - Integrated Development Environment
* IEEE - Institute of Electrical and Electronics Engineers
* N/A - Not Applicable
* UI - User Interface

### Document Structure

* IEEE: Later parts of the document will contain the plans for the structure of the team and communication within the team. Information regarding risk management, time estimates and how various problems will be handled can be found below.

## Project Organization

### 2.1 External Interfaces

* The most complex organizational boundary existing in this project is most likely communication with the professor. This is not of particular note as it is a typical student-professor relationship.
* Other organizations: Class Professor | Required for answers to questions regarding the project

### 2.2 Internal Structure

* Project progress is going to be following the Waterfall model due to its simplicity and effectiveness. Since our team is small, adapting to sudden changes (if necessary) shall be handled fairly well.
* The only interfaces within the development team are going to be simple communication with one another. We will not have a “manager” per se so much as there will be a leader that will be organizing as well as actually working on the project like everyone else. With everybody operating on the same social level we remove any complex interfaces from the project.
* Communication with the professor is generally done via email or questions asked whilst in class. In the worst case scenario there will almost always be two opportunities a week to get in contact with the professor. In reality, access to the professor is fairly open any time of the week. This should not be a major barrier to the completion of the project.

#### **Table 2 - Job assignments for each group member**

| Name | Project Role |
| --- | --- |
| Cameron | Project Manager | Backend Programming | Assist anyone that needs more help |
| Owen | UI Programming | Assist anyone that needs more help |
| Thomas | Front and Backend Programming | Assist anyone that needs more help |
| David | Technical Documentation | Programming | Assist anyone that needs more help |
| Ethan | Technical Documentation | Programming | Assist anyone that needs more help |

#### 

### 2.3 Roles and Responsibilities

* Various aspects of the project are going to be available for group members to work on. Everyone is going to work on something every week, but we want each member to choose their desired work if possible. Thus, we will give each other the option to work on one of various things that needs to be done in a week. The list of work activities that this ideology applies to are as follows. At the beginning of each week there will be a meeting held within the team where team members will pick what work they would like to be doing that week. This is also where any issues can be hashed out regarding that week's work schedule.
  + Ordering system framework
  + Meal and ingredient building tools
  + Detailed aspects of the ordering system
  + Visual design of the UI
  + Order backup system
  + Convenience features
  + Cleaning up of code documentation
* Some parts of the project are things that everyone will have to do at some point. Those are as follows
  + Bug fixes
  + Testing
  + Code documentation
  + Polishing
* Our project ideology will ideally give members a sense of completion and purpose to the project by allowing them to choose what they are contributing on a weekly basis. Unfortunately, people will not always be able to work on what they want to work on. Again, we are relying on our small group size to assist us in being able to hold one another accountable for taking responsibilities and completing those responsibilities in a timely fashion.

## Managerial Process Plans

This section of the Project Management Plan specifies the management processes for the project. This section defines the plans for project start-up, risk management, project work, project tracking and project close-out.

### 3.1 Start-up Plan

### **3.1.1 Estimates**

* It is estimated that 5 hours of coding each week will be done collectively, this is made with a confidence level of 3 out of 5. Each student will need access to a computer to work on code and documentation. There are no costs associated with the project assuming everyone has access to a computer.
* It is assumed that most students in this class have taken at least one coding class and either are familiar with Java or can quickly catch on. The coding portion however is not expected to be the most time consuming portion of the project. It is predicted that the documentation will take up the majority of the time.
* This estimate is based on previous coding projects that have happened at Kettering or a work environment. Most of the time planning for the project and making sure that documentation is in place takes longer than actually writing the code.
* In the event that the time estimation is incorrect, the new estimation will be based off of what was able to be done in the originally allotted amount of time.
* Re-estimation will be done on a weekly basis to make sure that all aspects of the project are making adequate progress.

### **3.1.2 Staffing**

* Specify the number of required staff, providing the following details:
* 5 workers with moderate skills in documentation and Java programming
* 11 weeks or 1 semester at Kettering.
* The staff will consist of the students in Group 3.
* A Gantt chart or excel sheet will be used to track progress on the project.

### **3.1.3 Project Staff Training**

* Each member will have experience from at least one coding class.
* Specify the following training information:
* Java training as necessary.
* Preferably no workers require training although up to 5 can be trained.
* The training provided will be in a lecture/hands on fashion.
* No other training besides possible Java programming is required.

### 3.2 Work Plan

### **3.2.1 Work Breakdown Structure**

* Work activities are but not limited to documentation and Java development.
* Specify the following factors for each work activity:

#### **Table 3 - Work Breakdown Structure**

| Major | Code Documentation | Coding | Testing | Leading |
| --- | --- | --- | --- | --- |
| Minor | Document/comment code. Update relevant README’s and project documents that have been submitted as deliverables based on what is going on during the coding process. | Working on the various elements that make up the project. These include but are not limited to the backup system, the ordering system, the GUI and various small features that will be included. This also includes bug fixes. | This will involve going through the code and testing sections for bugs and other problems. If there is time this will include writing unit tests to further ease maintainability. | This will involve keeping the rest of the group on schedule, dealing with any group issues that arise and working on ALL other aspects of the project. |

* The level of decomposition internally within the WBS may vary depending on the quality of the requirements, familiarity of the work, applicable level of technology, etc.

### **3.2.2 Schedule Allocation**

* Around 3 hours of the week are dedicated to the documentation and 2 hours are dedicated to Java development.
* It is critical to stay on top of current assignments and not fall behind.
* Possible constraints on the schedule would be due to an increase in workload at school.
* Some milestones include a working ordering system and the associated documentation.
* Staff member availability will be documented on an excel spreadsheet.

### **3.2.3 Resource Allocation**

* Each work activity will be resulting from working on the documentation or working on the Java code to be used in the project.
* There will be 5 staff members working on the Java development project and documentation.
* Specify, as appropriate, the allocation of the following resources:
* 5 staff members total
* A computer and active internet connection
* Java runtime environment
* No special testing allotted
* Support from teammates and professor.

### **3.2.4 Budget Allocation**

* There is a budget of 0 dollars for the project.
* The estimated costs for activity, personnel are as follows:
* No travel is expected
* All meetings will be done virtually or online
* All staff members are expected to supply their own computers,
* Java is available for free
* No simulation or special testing is expected
* No additional administrative support is required

### 3.3 Project Tracking Plan

### **3.3.1 Requirements Management**

* If changes to the requirements are made they will be updated in the progress tracking spreadsheet.
* If changes are made the due dates and time allotted for certain portions of the project might change. This will require the project schedule to be reworked.
* To limit the number of changes the group will meet and decide if a change is needed or a work around exists.
* The code and documentation will be constantly updated and validated to make sure the requirements are being met.

### **3.3.2 Schedule Control**

* For the schedule control activities we have the following:
* A spreadsheet will be used to monitor progress.
* If there is an issue on meeting deadlines changes will be made so that the team as a whole is not stuck
* A shared spreadsheet will be used to track progress and make sure that satisfactory progress is being made..
* The objective criteria will be that each portion of the program works, for example, the UI and the .txt database are correctly working together.

### **3.3.4 Quality Control**

* In order to ensure the quality of our application, every team member will not only review and test their own code, but will be responsible for checking over other team members' work as well. In addition to this, we will thoroughly test our program so that the final version is as polished as possible. We will also be meeting with the professor each class to ensure the quality of our documents. This means that each week we should have input from 5 internal sources and 1 external source.
* To measure the overall quality of our work we will be using our grade in the course, as well as our general sense of quality.
* If a team member appears to be struggling with the work processes, we may work with them to see where the issue is and try to find a solution that works for everyone. In terms of the verification and validation process, we will be using thorough testing of the code to ensure its quality. We will also be reviewing the documents internally before delivering them, as well as meeting with the professor at least one time prior to submission.

### **3.3.5 Reporting**

* Documentation will be done in accordance with templates uploaded to blackboard. Reporting of issues or advancements within the team will be reported on Discord. Issues regarding the team will be reported to the professor via email.
* Discord and email will be used as main means of communication.
* Communication with the group will happen once a week at a minimum.

### **3.3.6 Project Metrics**

* Metrics will be collected in an excel spreadsheet that will also allow for project advancement tracking.
* Specify the following metrics process information:
* Lines of code written and pages of documentation written.
* Collected once a week
* The metrics will be reported in a shared excel spreadsheet

### 3.4 Risk Management Plan

* Risk management will be conducted via team discussions in Discord, tracking any changes that might occur.
* Initial risk factors will be identified when creating the project plan.
* Describe the following:
* No contingency planning will be done as this is for a school project.
* Risk management activities consist of making sure the project and documentation are proceeding as expected.
* Risk management will be performed on a bi-weekly basis.
* All team members are expected to seek out risks and report any findings
* Any risks will be disclosed in Discord when they are discovered, if the professor needs to be notified it will be done in email.
* Identify and describe the applicable impact of any of the following risk factors:

| **Risk** | **Impact** | **Probability** | **Contingency Plan** |
| --- | --- | --- | --- |
| Customer Project Relationship | Medium | Highly Unlikely | Given the “customer” in this case has already approved of the idea we pitched we should be in the clear as long as we are able to implement all of the features that we said we would. |
| Contractual Risk | Large | Highly Unlikely | If we run into contractual issues the best thing we can do is negotiate with the customer. In the event negotiations fall through the project's budget is $0 so the only thing we lost was our time and effort. |
| Program Failure | Large | Highly Unlikely | In the event the program fails the restaurant using it they would be forced to take orders manually until the problem could be rectified. |
| Failure to Complete | Large | Possible | In the event the program cannot be completed before the due date there will have to be negotiations with the customer (in this case the professor). Whether or not things work out neatly in this situation will be mostly up to the customer. |
| Environment Problems | Small | Probable | There is a chance that there will be a customer that will be unable to use the program due to it being created in Java. The solution to this is to simply install Java on their system. If this is not possible then this solution will have to be reevaluated. |
| Staff Failure | Medium | Possible | There is a chance that group members will not be able to complete their assigned work in a given week. If this happens the solution is to assign them more work the following week and expect them to make up the work that was not completed the previous week. If that work begins to pile up dangerously a group member that has less work that week will step in and assist the struggling member. |
| Budget | Small | Highly Unlikely | In the event the budget is exceeded the group will simply have to discuss how to split the cost. |
| Customer Dissatisfaction | High | Possible | In the event the customer does not accept the deliverables there will simply need to be negotiations to understand what it is that the customer wants and a meeting to decide whether or not it is worth it to fulfill the customers wishes. |

* There is a low risk for the customer-project relationship, the product is built to make the customers life easier.
* There is a medium risk for contractual risks, we are currently in hard times for the restaurant business.
* There is a medium risk the project will be accountable for a technological risk. One scenario is if the ordering system does not work properly, taking orders and payments would have to be done manually.
* There is a low risk the project will fail due to complexity reasons.
* There is a low risk that the project will not work in other environments since it is written in Java.
* There is a low risk that staff members will not be able to complete expected duties.
* There is low risk of exceeding the budget due to the budget being none.
* There is a medium risk the customer will not accept the deliverables. There will be contact with the customer although prototyping will not be done.

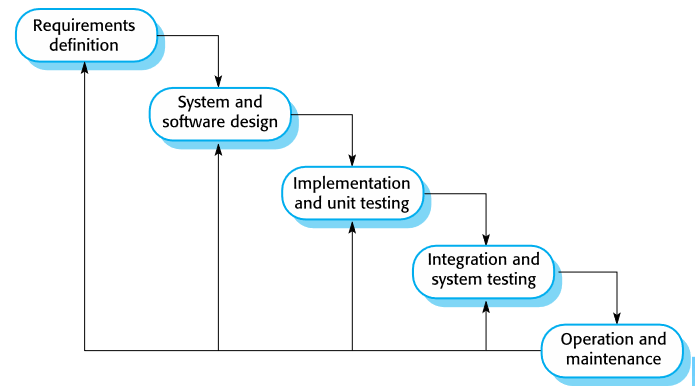
### 3.5 Project Closeout Plan

* Two weeks before the project is due, the group will meet and discuss closing out the project.
* This will most likely involve discussing any last minute features and whether or not group members are content with the quality of work being provided. If there is any discontent with that there will still be two weeks to rectify the problems.

## 4. Technical Process Plans

### 4.1 Process Model

* We intend to use the waterfall model to track our development of the ordering system, as seen in Figure 2.



#### **Figure 2 - Format of the waterfall project management plan**

* The team members shall communicate any relevant project information with each other via Discord and email. The code and documentation of this project will exist in a GitHub repository that is publicly accessible.
* The timing of our work is intended to be in accordance with due dates assigned by the instructor of this course. Per the waterfall model and the constraint of Kettering University’s term length, it makes sense to evenly divide the phases between the weeks of instruction that we have. Following the conclusion of this document, we will have taken three weeks to complete the “Requirements Definition” phase. Our remaining time of this project shall be split between the remaining phases. We plan to allot one week for “System and Software Design”, up to three weeks for “Implementation and Unit Testing”, up to two weeks for “Integration and System Testing”, and a final week for “Operation and Maintenance.” The actual time we take to complete these phases may be shorter, and thus we may have additional time in the later phases of the waterfall model. Ideally, we shall not need to return to previous phases of the model.
* Due to our small team size, all team members are expected to review the work that is collaboratively created and submitted. The process of reviewing work will be continuous until project completion, but scheduled reviews may be organized if necessary.
* Major milestones of this project include but are not limited to:
* Completed outline of Java classes with clearly defined desired functionality
* Completed visual design of the UI
* Successful implementation of editable food ordering and meal shortcuts
* Seamless deployment of the software on new machines
* Completed testing and debugging of software issues
* Completion of project polishing
* Project delivered
* We seek to satisfy the following baseline requirements:
* The user interface shall be properly threaded, meaning it should not visually or functionally hold up at any stage.
* The user interface should be visually consistent and simple to navigate.
* The user interface must not require unnecessary steps to toggle features.
* Features should behave consistently no matter the food ordered.
* The project deliverables include an executable program, the code that drives the program, instructions on how to use the tool and documentation for the code that drives the tool.
* The team’s work is periodically reviewed by the course instructor; project-planning documents are critiqued by the instructor and changes are then suggested to the group.
* Project initiation began upon a random group member assignment by the course instructor. Project ideas were then considered among the group members. The restaurant ordering system was collectively decided to be a challenging enough project to span Kettering University’s term length and also provide enough work for all members. Upon project completion, all deliverables will be provided to the instructor and the group members will display their project in use for the class.

### 4.2 Methods, Tools, and Techniques

* Our development of the software will be written in Java programming language. Our coding conventions will follow that of typical Java programming projects, as taught in previous Kettering University courses. The visual design of the UI (before its actual completion) shall be done in some image editing software such as GIMP. We will choose one of the many existing Java GUI frameworks to develop the UI. The team members will be using their IDE of choice while developing the program.
* In order to identify issues within the software, extensive testing will be done with regard to different order combinations, item cancellation, repeated toggling of features. At times this testing will include direct intent to break the program. Known issues will be tracked in our internal communications via Discord.
* Code documentation will be written in some basic text format such as Microsoft Word or GitHub’s inherent README file creation.
* The project deliverables will be provided to the instructor via Blackboard submission.

### 4.3 Infrastructure

* Team members are expected to use their personal computers for project development. If accommodations must be made, the members may also utilize Kettering University’s library resources.
* While the team members may be using different operating systems and IDEs during the development of the project, the end-user is assumed to be using a Windows machine that may run Java programs.
* The group members may meet physically on campus as well as virtually from their homes. Meetings with the instructor may be done in class, during instructor office hours, and virtually.
* All submitted work shall be in accordance with Kettering University’s policies as well as the CS-471 course syllabus, including but not limited to those concerning academic integrity and ethics.

### 4.4 Product Acceptance

* End-user acceptance of the deliverables will be based on the instructors grading rubrics for our various submissions.
* The project shall conclude as a fully tested, reliable, and cohesive GUI for managing food orders. The UI will be deemed acceptable if it is properly threaded, works without unnecessary steps to perform simple actions, and behaves consistently. The functionality of the UI for managing orders will be acceptable if orders may be changed in any possible manner and still behave correctly, orders are logged for unforeseeable circumstances, and orders are not constrained by number of items ordered or the time it takes to order them. The code documentation will be acceptable if all functions are clearly defined and explained, all known bugs (if any) are noted, and there are no grammatical or spelling errors. The document that explains how to use the ordering system will be acceptable if steps are written concisely and there are no grammatical or spelling errors.
* No formal agreement of the above criteria was made between the group members and the instructor.
* A live demonstration of the food ordering system will be shown off to the instructor and classmates of the group members.

## 5. Supporting Process Plans

### 5.1 Documentation

* We will be preparing several documents to be delivered. Those documents are:
  + Project Proposal
  + Project Management Plan (SPMP)
  + Project Requirements (SRS)
  + Midterm Status Report
  + Project Design (SDS)
  + Project Test Plan (STS)
  + Project README file
* We will be using the template provided by the professor as well as following the IEEE standards.
* All group members will work together to complete and prepare each of the documents.
* All group members will also share the responsibility to review each document. We will also be receiving feedback from the professor on each of the documents.

## 6. Additional Plans

* In order to meet security requirements we will only be using libraries that have a long roadmap of future maintenance to ensure that our application is secure
* A README will include installation instructions as our app will most likely not be the installation process that the average user is familiar with
* We have no plan for in person user training.
* While we will most likely not be maintaining this project if we ever were to update it we would most likely simply update the website distributing it and notify users they need to install the latest patch
* For product support our plan is mostly to just remain available for questions if the users have them



College of Sciences & Liberal Arts  
Department of Computer Science

**CS 471 Software Engineering**

**Winter 2022**

**Course Project**

# Software Requirements Specification

For

Restaurant Ordering System

Version 2.0

CS 471 - G3

Dr. Jamal Alhiyafi



1/31/2022

**Table of Contents**

[Revision History](#_gjdgxs) 3

1. [Introduction](#_1fob9te) 5

1.1 [Purpose](#_3znysh7) 5

1.2 [Scope](#_2et92p0) 5

1.3 [Definitions, Acronyms, and Abbreviations](#_tyjcwt) 5

1.4 [References](#_3dy6vkm) 6

2. [Overall description](#_1t3h5sf) 7

2.1 [Product Perspective](#_y7og8kyk2f3y) 7

2.2 [Product Functions](#_i6huji4ym1fx) 7

2.3 [User Characteristics](#_r4m2ol447jtg) 7

2.4 [Constraints](#_tc6ieklthxd9) 7

2.5 [Assumptions and dependencies](#_gnn6bj8idvdd) 8

3. [Specific requirements](#_ew83wxyzt1tb) 9

3.1 [Functional requirements](#_c8i8z8stxcdf) 9

3.2 [Performance requirements](#_59uapl2kbhx8) 11

3.3 [Design constraints](#_52zleg1682c0) 11

3.4 [Software system attributes](#_7p97czblose6) 11

4. [External Interface Requirements](#_lp63uw4n4yn6) 12

4.1 [Models for Program Execution](#_yma0mvgmtpgy) 12

## Revision History

| **Name** | **Date** | **Reason For Changes** | **Version** |
| --- | --- | --- | --- |
| Cameron Manning | 2/1/22 | Initial Fill Out | 1.0 |
| Ethan Knott | 2/3/22 | Initial Fill Out Section 2 | 1.0 |
| Owen Bickel | 2/7/22 | Revisioning & grammar changes | 1.0 |
| David Xia | 2/7/22 | Initial Fill Out Section 3 | 1.0 |
| Thomas Hayward | 2/9/22 | Filled out some more of Section 3 and added two GUI interfaces | 1.0 |
| Owen Bickel | 2/11/22 | Revisions after meeting with Dr. Alhiyafi | 1.0 |
| Owen Bickel | 2/11/22 | Created and added diagrams for functionality | 1.0 |
| David Xia | 2/22/22 | Reformatted document and fixed errors. Remade activity diagram. Edited sections according to feedback | 1.1 |
| Cameron Manning | 2/28/22 | Edited sections according to feedback | 1.2 |
| Ethan Knott | 3/17/22 | Table of contents numbering is now consistent, each section begins on a new page, fixed numbering in section 3, fixed indentation errors | 2.0 |
| Owen Bickel | 3/18/22 | Grammar and formatting changes | 2.0 |

**Table of Figures**

* [Figure 1 - Activity Diagram for Taking Customer Orders Through GUI](#_klv4ka90k0ps)

## Introduction

### Purpose

The purpose of this Software Requirements Specification is to create a document that the designers and developers can refer to when designing the software and making implementation decisions for the code as well as being a record of the business side of the development cycle was expected from the development team. The main audience for this document is the designers who will use it to make architectural design decisions.

### Scope

The software this document is documenting is being loosely referred to as “Restaurant Ordering System”.

The product is going to be a simple implementation of a restaurant ordering system that will be a cheaper option for small businesses. In further detail, it will be a GUI-based system where users will be able to start an order, select various items, select meals, modify items and essentially do everything necessary to translate a customer's order into something that a computer can understand. On top of this, there will be a backup feature that will save each completed order to a directory on the machine as a record of all the orders the machine has taken.

The goal of this software is to make something cheap and easy to deploy for small businesses. This is a huge benefit to the restaurant because they get simple, easy-to-maintain software that will not break often. Setup is also extremely easy which means owners will not have to spend a large amount of their time (or their businesses’ time) getting this software set up. Our goal for the software is to create it as a school project. At this moment no group members have voiced an interest in attempting to further develop the software and monetize it so there are no real goals outside of the previously mentioned one. Our objective for this project is to fulfill as many of the previously listed features as possible and finish in a timely manner. These objectives align with the goals of a school project.

The software needs to be relatively light on performance and must run on Windows 10 operating system.

### Definitions, Acronyms, and Abbreviations

* FR: functional requirement
* GUI: graphical user interface
* SDS: Software Design Specifications
* SPMP: Software Project Management Plans
* SRS: Software Requirements Specification

### References

* Manning, C., Knott, E., Xia, D., Bickel, O. and Hayward, T., 2022. *Project Proposal*. Undergraduate. Kettering University.
* Manning, C., Knott, E., Xia, D., Bickel, O. and Hayward, T., 2022. *Group 3 SPMP*. Undergraduate. Kettering University.

## Overall Description

### Product Perspective

* This system is independent and totally self-contained
* The only dependencies it has are on the computer it runs on and the language it is coded in

### Product Functions

The functions and subfunctions of the product are as follows:

* Create customer order with GUI
  + Add a food item to an order
  + Remove ingredients from food items
  + Delete items from meals if undesired
  + Add a la carte items (sides and drinks)
* Log previous orders
* Cash-out customers

### User Characteristics

The expectation of the users are as follows:

* Proficient in the language orders are taken in to ensure that orders are taken correctly and efficiently. In this case, the GUI will be developed using the English language.
* Low technical experience, the ordering system is designed to be simple to use and will require minimal training.
* Moderate restaurant/cash register experience to ensure that customers are cashed out correctly.

### Constraints

The constraints of the project are as follows:

* No regulatory policies will apply to the ordering system. The ordering system does not handle sensitive data.
* There are no interfaces to other applications.
* There shall be no parallel functions as the program is extremely lightweight.
* The ordering system has a built-in audit system that logs every order taken and outputs it to a .txt file that can be checked if needed.
* Data validation should be done if any inputs are taken from the user to stop the program from crashing.
* There are no safety or security concerns as the program does not deal with sensitive data, nor should the program allow access to any sensitive data.
* Reliability is very important as the ordering system is a key component of a restaurant.
* A time constraint of about 5 weeks to code the project.

### Assumptions and dependencies

The following are the assumptions made about the user and their hardware.

* It is assumed that the machine running the ordering system will run a variation of Windows 10 with Java installed. The ordering system will be validated to run on Java 17.
* The user must have an input device whether it be a touch screen (recommended) or a mouse to navigate the system interface.

## Specific requirements

### Functional requirements

[Figure 1](#_klv4ka90k0ps) details the activity flow of the software when operated by a customer to order food Items using the GUI. For the core functionality, the software will contain the following classes of objects:

* + - GUI class

ID: FR 1

Title: GUI class

Reason: Interface for restaurant staff to interact with. This class in coordination with all other classes shall satisfy the functional requirement to take a customer’s order through the GUI.

Description: This class shall provide a simple and quick-to-use interface for the restaurant staff to take and complete orders with. This interface will allow for item selection, item deletion, and navigation to submenus.

Dependencies: The GUI class will be dependent on all other classes before it can be fully tested.

* + - FoodItem class

ID: FR 2

Title: FoodItem class

Reason: Class for defining FoodItem objects and the functions it needs.

Description: This class shall define an Item object that contains functionality such as obtaining item available & base price, returning a list of potential ingredients, ability to add and remove ingredients, and calculating a final Item price.

Dependencies: This class will not be dependent on any other class as it is one of the base classes.

* + - Meal class

ID: FR 3

Title: Meal class

Reason: Creates Meals that are a collection of Items

Description: This class shall allow multiple items to be formed into a single object. Through this class, items may be combined together, stored as a group, and calculated as a total meal price.

Dependencies: This class will be dependent on the item class.

* + - Order class

ID: FR 4

Title: Order class

Reason: Class for defining a Customer's order and functions required to track order information

Description: This class shall define an Order object that tracks Meal objects chosen by the customer. This Order object shall be ordered as an ArrayList of Meals. These Meals shall be removable from the order, a total price should be calculated, and a final order ID must be returned.

Dependencies: This class will depend on the item and meal classes.

* + - DatabaseOutput class

ID: FR 5

Title: Database class

Reason: Class for outputting saved or finished order to a text file.

Description: This class shall save all of the completed orders to a text file on the local machine.

Dependencies: This class will need the entire system to be working except for the GUI. It will need the meal class, the order class, the different options, and the ordering system.

* + - Ingredient class

ID: FR 6

Title: Ingredient class

Reason: Class for accessing assigned ingredients in the backend.

Description: This class shall give methods for setting and accessing ingredients for FoodItems.

Dependencies: This class will depend on the meal and item classes.

* + - IngredientlessItems Enum class

ID: FR 7

Title: Ingredientless items enum

Reason: This will provide an easy way to access all of the different ingredientless items and sauces.

Description: This class shall hold all of the different options that can be added or removed from items. The end-user of the product shall toggle radio buttons of the toppings that the customer wants.

Dependencies: This class will depend on the meal and item classes.

* + - GuiAPI class

ID: FR 8

Title: GUI API class

Reason: Backend side of the Menu

Description: This class shall be the main system running in the background. It is the basis for everything else and is what ties it all together.

Dependencies: This class will need all other classes except for the GUI as we could test it through the terminal. This class will depend on: the item, meal, order, database, and enum classes.

### Performance requirements

The ordering system must be fast and responsive. We are defining this requirement as reactive to end-user input within 500ms, and flexible support for at least four meal options and ten food items. Selecting various permutations of meal options, topping options, and performing additions and removals should all maintain expected functionality.

### Design constraints

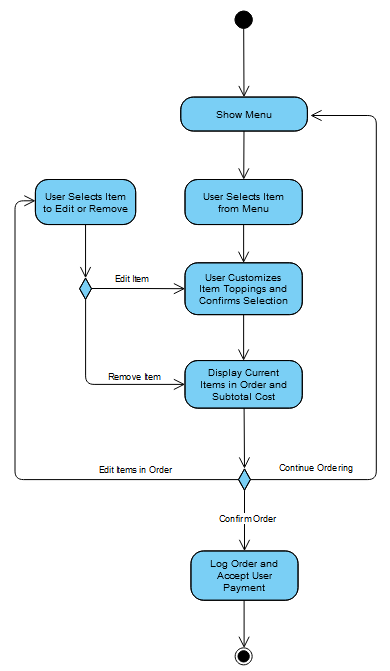
The ordering system will only support one end-user, as no user authentication will be required. The logging of previously ordered items will be limited by hard drive space.

### Software system attributes

* + - The ordering system will require a download of Java. Also, Windows 10 must function as expected for the system to perform well.
    - The system does not require any specific treatment for security or maintenance. Its ease of use shall be contingent on a rigidly designed GUI.

## External Interface Requirements

### Models for Program Execution



#### **Figure 1 - Activity Diagram for Taking Customer Orders Through GUI**

This figure represents the process by which employees will take customer orders in the system.

The above figure is mapped to FR 1



College of Sciences & Liberal Arts  
Department of Computer Science

**CS 471 Software Engineering**

**Winter 2022**

**Course Project**

# Software Design Specifications

For

Restaurant Ordering System

Version 2.0

CS 471 - G3

Dr. Jamal Alhiyafi



2/15/2022

This Software Design Specification was prepared and provided as a deliverable for CS-471, Winter 2022, and it will be used by the development team.

This document is based in part on the IEEE Recommended Practice for Software Design Descriptions.

| Name | Email |
| --- | --- |
| Owen Bickel | bick7185@kettering.edu |
| Thomas Hayward | hayw6218@kettering.edu |
| Ethan Knott | knot1657@kettering.edu |
| Cameron Manning | mann6441@kettering.edu |
| David Xia | xia0627@kettering.edu |

**Table of Contents**

[Software Design Specifications](#_72kqkn1svzoo) **1**

[Revision History](#_d2ejpkrt9rsz) 5

[Introduction](#_lf7o8fb4fdgj) 8

[Purpose](#_6syqi2igjrd4) 8

[Scope](#_fghf4qhoz1un) 8

[Definitions, Acronyms, and Abbreviations](#_zbdmnpjnwltz) 8

[References](#_6bz2jvou1gnr) 8

[System Overview](#_rmlq6r8rb426) 9

[Design Considerations](#_xdhr6c5rmhnn) 10

[Assumptions and Dependencies](#_j8a7iz6vj8c6) 10

[General Constraints](#_x7zwqkutip9u) 10

[User Interface Design](#_d37npcx80yyy) 11

[Overview of User Interface](#_u2vcft4c360k) 11

[Interface Design Rules](#_c1tg5aira564) 11

[Screen Images](#_2r87gk1a9bty) 12

[Screen Objects and Actions](#_hf8fj0j8n0f4) 13

[System Architecture](#_h1vmq3n9ar2s) 15

[Architectural Design Approach](#_ugohj6mrggx4) 15

[Architectural Design](#_5up8uxi3xo59) 16

[Data Design](#_64e1feud9qlk) 18

[Data Description](#_qfc6cizih7dk) 18

[Data Dictionary](#_ghsqaj1rpeux) 18

[Database Description](#_am7f5h7lcten) 19

[Detailed System Design](#_ht5p7tvbbfbm) 20

[Other Design Features](#_kxu6yjpo0ync) 26

[Requirements Traceability Matrix](#_idkfeyd3i61k) 27

## Revision History

| **Name** | **Date** | **Reason For Changes** | **Version** |
| --- | --- | --- | --- |
| Cameron Manning | 2/14/22 | Initial fill out | 1.0 |
| Ethan Knott | 2/17/22 | Initial fill out | 1.0 |
| Thomas Hayward | 2/21/22 | Initial fill out | 1.0 |
| Ethan Knott | 2/23/22 | Filled out more | 1.0 |
| Owen Bickel | 2/23/22 | Initial fill out of Interface Design Rules, Screen Objects and Actions | 1.0 |
| David Xia | 2/24/22 | Section 5 Initial fill out. | 1.0 |
| Owen Bickel | 2/25/22 | Added to 5.1 and 5.2. Made slight revisions across the SDS, including figure titles | 1.0 |
| Ethan Knott | 3/17/22 | Fixed numbering under section 7, table of tables now includes page numbers, page numbers added for table of figures. Added diagrams for important classes in section 7. | 2.0 |
| Owen Bickel | 3/18/22 | Grammar errors and formatting inconsistencies. | 2.0 |

**Table of Tables**

[Table 1 | Data Dictionary](#_g1kr9uabrcwx) 13

[Table 2 | Functional Requirements -> Classes](#_xu78hf5u2q3r) 14

**Table of Figures**

[Figure 1 | Home screen of the ordering system](#_mrrcd21g4cpq) 5

[Figure 2 | Customization menu for different items](#_jtp8q6myvyol) 5

[Figure 3 | Organization of the Model-View-Controller Architecture](#_l9kf86gma95b) 8

[Figure 4 | Activity diagram depicting the flow of the Ordering System](#_9olclcp2aec) 10

[Figure 5 | Activity diagram depicting the flow of the Ordering System](#_9olclcp2aec) 10

[Figure 6 | Activity diagram depicting the flow of the Ordering System](#_9olclcp2aec) 10

[Figure 7 | Activity diagram depicting the flow of the Ordering System](#_9olclcp2aec) 10

## Introduction

### Purpose

The purpose of this SDS is to create a document that the design team can closely follow for the implementation process. It will also serve as a record of what the design team envisioned the product to look like. The main audience for this document is the development team but can be shared to other stakeholders to explain how the project is progressing.

### Scope

The software this document is documenting is being loosely referred to as “Restaurant Ordering System”.

The software being developed should be a simple implementation of an ordering system that can be cheaply bought by small restaurants. There will be an easy-to-understand GUI that enables a user to fully complete an order: starting an order, selecting their items or meals, customizing said items and meals, and checking out. After the order has been placed, there should be a system that captures the order and stores it in a database that is local to the machine.

The objective of this product is to create an affordable ordering system that small businesses can use to boost their business. The ordering system should be extremely easy to set up and use, giving the restaurant more time to focus on satisfying customers.

### Definitions, Acronyms, and Abbreviations

* FR: Functional Requirement
* GUI: Graphical User Interface
* GB: Gigabyte
* MVC: Model-View-Controller
* PC: Personal Computer
* SDS: Software Design Specification
* SPMP: Software Project Management Plan
* SRS: Software Requirements Specification
* UI: User Interface

### References

* Alhiyafi, J., (2022). *Architectural Design* [Powerpoint slides]. College of Sciences & Liberal Arts, Kettering University.
* Manning, C., Knott, E., Xia, D., Bickel, O. and Hayward, T., 2022. *Project Proposal*. Undergraduate. Kettering University.
* Manning, C., Knott, E., Xia, D., Bickel, O. and Hayward, T., 2022. *Group 3 SPMP*. Undergraduate. Kettering University.
* Manning, C., Knott, E., Xia, D., Bickel, O. and Hayward, T., 2022 *Group 3 SRS*. Undergraduate Kettering University.

## System Overview

The ordering system will be composed of 8 class files that will work together to create the ordering system. The GUI class will be responsible for creating the GUI system that the user uses to interact with the ordering system. The Item class will hold item availability, price, and have the option to remove and add ingredients. The Meal class will allow for multiple items to be grouped together and stored as a group as well as calculate the new price. The Order class allows for multiple meals to be grouped together and tracked as one. The Database class allows for orders to be exported and saved to the local machine. The Enum class allows for the storage of options that apply to various food types, you will be able to toggle which toppings are applied via a checkbox. Finally, the Ordering System class will tie all the other classes together and has dependencies on the previously listed classes.

## Design Considerations

### Assumptions and Dependencies

The ordering system has the following dependencies and assumptions needed for operation:

* No training is required on how to use the ordering system.
* A display for the ordering system to be displayed on.
* No planned updates in functionality.
* It is assumed that the machine running the ordering system will run a variation of Windows 10 with Java installed. The ordering system will be validated to run on Java 17.
* The user must have an input device whether it be a touch screen (recommended) or a mouse to navigate the system interface.

### General Constraints

* Operating system must be Windows 10.
* Java must be installed.
* A display and input device are required.
* No standards are required to be followed.
* No security restrictions.
* Memory limit of 1GB.
* No network communication required.

## User Interface Design

### Overview of User Interface

The user should be able to perform all of the actions required to start, customize, and complete an order. Most of the functionality should be present in the main screen to keep the system compact, concise, and easy to understand. From the main screen, the user should be able to add any of the available items or meals to their order. When adding an item or meal that is customizable, there should be another menu that appears showing the different options for that particular item, as well as showing the final price of the item. This menu will also be accessible from the home screen via a button. The home screen should also contain a “Summary” portion that will display all current items in the order and should be interactable, allowing the user to select an item and then customize or delete it. Once the user is finished with their order, there will be a button that takes them to the payment screen. Here, the user can select cash or card, as well as go back to the order.

### Interface Design Rules

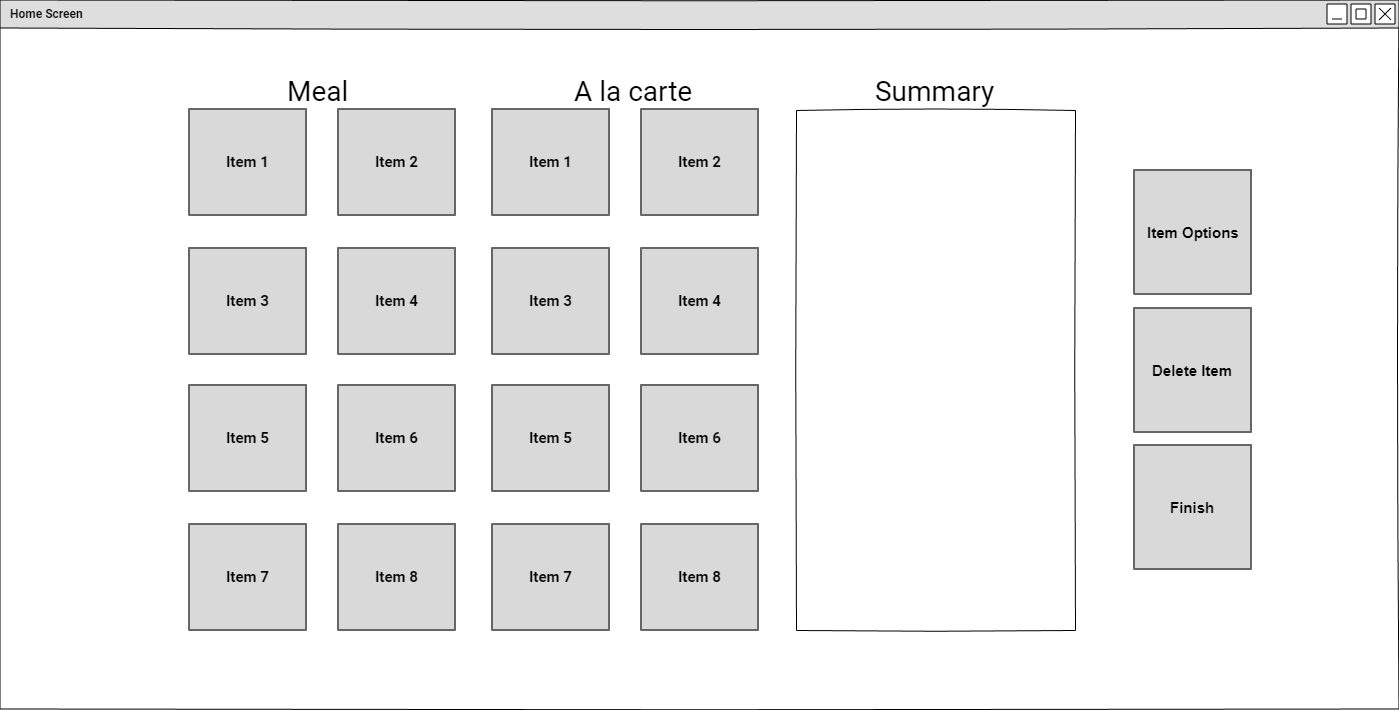
The interface shall be visually simple, and linearly navigable. When secondary windows are opened, for example, the Item Options menu, they should receive priority over the home screen of the GUI. If the user clicks in the home screen window while a secondary screen is active, then no action should be taken.

All windows shall be scalable to some degree. We intend to maintain a minimum window size of 1920x1080 so the GUI elements are not visually displaced when scaled.

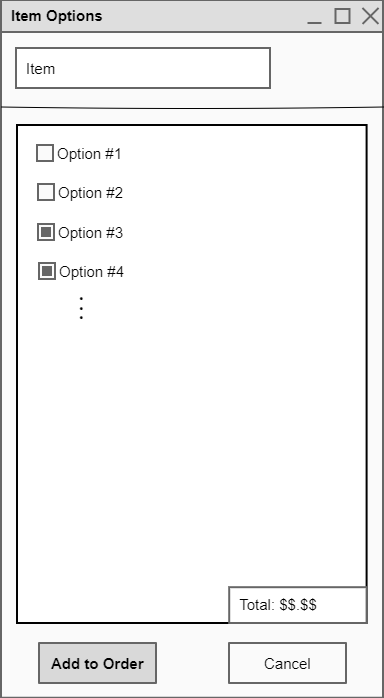
Users shall be able to deselect entire side items from meals. Other side items should be selectable for replacement in the meal.

Orders of buttons in menus should be consistent (for example, the order of confirming a change and discarding a change). All button fonts and styles shall be similar, if not the same, to avoid visual confusion.

### Screen Images



#### Figure 1 | Home screen of the ordering system



#### Figure 2 | Customization menu for different items

### Screen Objects and Actions

On the home screen, the following screen objects, and associated actions will be displayed and carried out:

* Minimize Window Button
  + Minimizes the home screen to the PC taskbar.
* Maximize Window Button
  + Make the window fullscreen.
* Exit Application Button
  + Closes the application window, prompting the user whether they want to continue or abandon the current order.
* Meal Column Header
  + Denotes all meals, which automatically pair an entree item with a side item and drink.
* One button per meal item under the Meal tab
  + Pressing a meal button adds that entree and accompanying side item to the order, as well as a Cola for the default drink.
* A La Carte Column Header
  + Denotes all individual items to be sold a la carte. This may be a scrollable section with all items or a static menu with all side items and drinks.
* One button per individual item under the a la carte menu
  + Pressing an a la carte menu item button adds that item to the order.
* Summary Header
  + Denotes the summary box, which contains all items added to order with their respective customizations.
* Order Summary
  + Textbox of all items and their prices.
* Item Options Button
  + Opens the Item Options menu for whatever individual item is selected within the Order Summary. If the item is not customizable, then the menu will not be opened.
* Delete Item Button
  + Removes the selected item from the Order Summary and subtracts the item cost from the total cost.
* Finish Button
  + Back the Order up to a directory driven database.

The Item Options menu will have the following screen objects and respective behaviors:

* Minimize Window Button
  + Minimizes the secondary menu to the PC taskbar. This window will still have priority over the home screen.
* Maximize Window Button
  + Make the secondary menu fullscreen.
* Exit Application Button
  + Closes the Item Options menu, prompting the user whether they want to continue editing or abandon the current item customization.
* Item Name Textbox
  + Denotes the item that is being edited, whether that is a meal or individual item.
* One radio button per entree item topping
  + Selecting an unchecked radio button adds the customization option to the item and sums the custom option price with the current item total price.
  + Deselecting a checked radio button removes the customization option from the item and subtracts the custom option price from the current item total price.
* Save Changes Button
  + Confirms changes to the customized item and recalculates item total for total order total price.
* Cancel Button
  + Cancels item customization choices that were made since the window was opened.

## System Architecture

The software will follow the Model-View-Controller (MVC) architecture pattern, which divides the software into three main components:

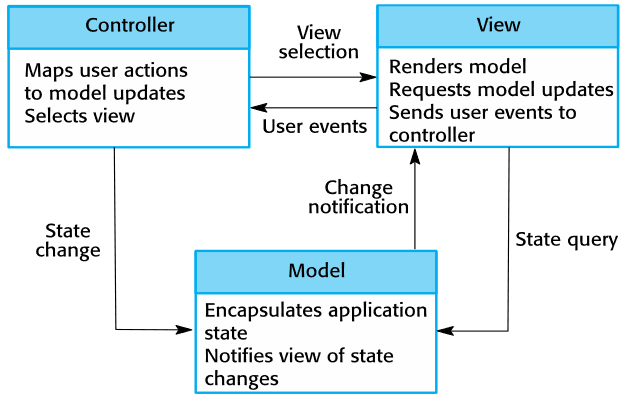
* The View or Graphical User Interface
* The Model which captures the state of the customer order
* The Controller which controls interactions between the user and the Model.

### Architectural Design Approach

The Model-View-Controller is a popular architecture pattern used in web applications and is adopted in many software frameworks, so its organizational effectiveness is proven and reliable. The simplicity of the MVC is also very suitable for the Restaurant Ordering System which does not have a lot of parts required for its operation. Dividing the software into more components with a more complex architectural structure is not necessary and would overcomplicate the design.

Using MVC, the software is partitioned into three elements each with distinct and easily comprehensible roles and responsibilities, so the developers have a clear vision of what each element needs to do. The interaction between the three elements of the software is also well defined so the developers can easily refer to it.

The typical organizational structure of the Model-View-Controller is shown here:



#### Figure 3 | Organization of the Model-View-Controller Architecture

### Architectural Design

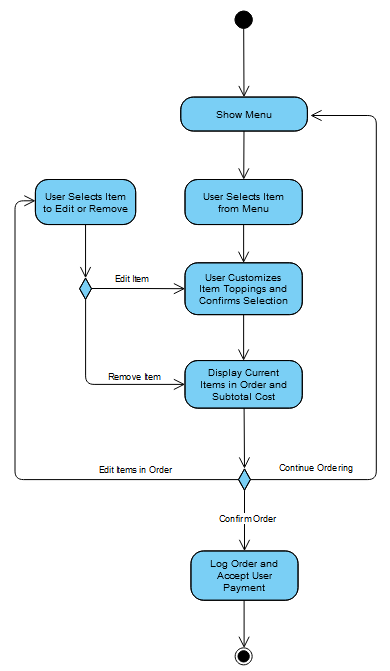
According to the MVC architecture pattern, there are three interconnected modules for the Restaurant Ordering System: the Model, View, and Controller.

The Model module is composed of all the Classes for food items and the Order Class. These Classes allow the Restaurant Ordering System to keep an internal model of the state of the customer's current order, such as keeping track of what items are ordered and how each item was customized. The functions defined inside this module are primarily concerned with storing, editing, and giving information.

The View module is composed of the GUI Class. This Class is focused on displaying information to the user and accepting user input. This module must facilitate the activity flow shown in Figure 5 by appropriately displaying different menus and buttons with each step of the diagram. The View module pulls information from the Model module. When the GUI accepts user input to make changes to their order, it will activate functions from the Controller module to affect the Model.

The Controller module is composed primarily of the Order Class and other backend Classes. This module is responsible for manipulating the Model module, such as adding, removing, or customizing food items from the customer order. This module is also responsible for logging customer orders. Most of the functions defined in the Controller module are activated in response to user input from the View module.

Because of the simplicity of the software, the Order class does take on some of the responsibilities of both the Model module and the Controller module as it stores the information of the current food order and also takes user input from the View module to manipulate that information.



#### Figure 4 | Activity diagram depicting the flow of the Ordering System

## Data Design

### Data Description

There is a text file that the ordering system outputs completed orders to. This database is used as a log of what orders have been taken. The ordering system utilizes Arraylists. The base of all the data structures is found in the FoodItem, Meal, and Ingredient classes. These provide the fields that are used by complex classes, such as the ingredientList which is built upon Ingredient. These classes all provide methods that can be called upon to retrieve data about them, such as a getFoodName() method used in the FoodItem class.

### Data Dictionary

### Table 1 | Data Dictionary

| **Variable Name** | **Variable Type** | **Class** | **Description** |
| --- | --- | --- | --- |
| foodList | ArrayList<FoodItem> | Meal | Contains all of the food items that are part of a meal. |
| foodName | String | FoodItem | Provides the full name of a food item. It will be what the users see. |
| ID | Int | FoodItem | Provides the ID of a food item. It will be what gets used in the backend. |
| ingredient | String | Ingredient | Provides the name of the ingredient . |
| IngredientlessItems | Enum | IngredientlessItems | Contains all of the items that are not made up of individual .ingredients. |
| ingredientList | ArrayList<Ingredient> | FoodItem | Provides a list that contains all of the available ingredients. |
| mealName | String | Meal | Provides the name of a meal that the user will see. |
| order | ArrayList<FoodItem> | Order | Contains all of the items currently added to the order. |
| order | Order | GuiAPI | Contains all of the necessary information regarding orders for the GUI to use. |
| price | Int | FoodItem | The price of a food item. |

### Database Description

Our database will be a simple directory that is stored on the computer running the software. In this directory, previous orders will be saved as a .txt file with a meaningful name. The name should include the date and time it was placed, as well as abbreviations of the items or meals that are a part of the order.

## Detailed System Design

Most components described in the System Architecture section will require a more detailed discussion. Other lower-level components and subcomponents may need to be described as well. Each subsection of this section will refer to or contain a detailed description of a system software component. The discussion provided should cover the following software component attributes:

1. **GUI Class**

A.1: This is the Java class in our project that is going to handle generating the GUI that the users will be clicking on. Every part of the project’s GUI is going to be managed in this class

A.2: The primary responsibility of this class is to manage the GUI of the entire project. It will manage the connection between the user and the computer.   
A.3: This class does have a bit of a time constraint. One of the requirements for the project was that any input from the user should result in SOME response from the system within half a second. Given the GUI class is going to be the most expensive class (processing wise) in the project, it is important that the way it is implemented does not result in slow responses to the user.

A.4: This class does not have subclasses per se but it does have a few different processes that will need to be implemented. First, when the system starts there will be a large screen that includes meals and individual items. This menu will include the ability to add items to an order, edit those items, delete an item from the order and also presents the ability to add lots of items at once via a meal. Whenever an item is edited a new menu will pop up that includes the various parts of a food item that can be removed. Users will be able to click on things that they do not want on the food.

A.5: This class is going to be directly used by the back end of the project. What that means is that the part of the project that runs the ordering processes and behaviors on the back end is what is going to be directly working with the GUI. This class is going to heavily use the backend object classes and the backend areas of the project that create the flow of an order.

A.6: This class will not have any threading issues and is only reliant on a screen being connected to the computer that our software will be running on. On top of that it will be wholly reliant upon a Java GUI library (Swing) that will be used to simplify the development process. On top of this it will be reliant on every other class in the project with the exception of the database class.

A.7: This class is going to be taking advantage of Swing, a prebuilt Java library, to generate the GUI for the project. By taking advantage of these libraries this class ought to be remarkably simple implementation wise. The goal of this class will be to implement the figures showing our plan for the front end. This class will get data and information to populate the front end from the classes that control the data classes via methods that will be present there.

A.8: This class provides no service to the rest of the project. It is merely the connection to the user.

A.9 This can be shown further by the activity diagram shown in figure 4.

1. **Item Class**

B.1: This is the data class that is going to store the information contained by an item on the menu. This could include anything from the price of an item to the various ingredients an item contains.

B.2: This class allows us to create food items programmatically from an object-oriented perspective. By allocating a file to a food item we ease the complexity of implementing the backend. This class will be used to store information about various ingredients and it will be shuffled around in the back end of the project to mimic the orders being created on the front end. The services this class provides are essentially being a data object that stores information on the various food items that will have to be added to the project.

B.3: There will be no constraints on this class. It is an extremely simple Java class that will not pose any kind of violation of the project requirements.

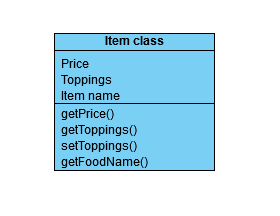
B.4: There are not any major subsections of this class. It could be said that the individual data objects making this class up (most likely individual strings and ints) might qualify as subsections.

B.5: This class is going to be used as data storage that can be used to populate the front end as well as a data object that will get added to orders in the back end as a user inputs an order on the front end. This class is extremely simple. It will not be reliant on any other classes in the project to exist and work as intended on its own. The aforementioned GUI will most likely have calls to this class to pull information for the front end and the ordering system class will also be creating orders using the item class as components. On top of that, the meal class and order class will also be storing these classes as base data that can be pulled by the back and front end. This class will also be involved with the database class.

B.6: This class will have little to nothing else that it will rely on.

B.7: All this class needs to do on a technical level is store the name of the item, the item's price and the ingredients that make up the item. There will be getters and setters in this class that will ease the process of getting that information out of this class.

B.8: This is simply a data storage class. It will be used to pull data in pretty much every class in the project. All other classes are dependent on the functionality of this one, which is why it is going to be implemented as simply as it is.



#### Figure 5 | Class diagram depicting the Item class

1. **Meal Class**

C.1: This class will store a list of items (the item classes) that will make up one of the meals that the user can quickly select on the front end.

C.2: This class is going to function as a list of the items that make up a meal. It will also have getter methods that can be used to retrieve those ingredients however the front/back end requires.

C.3: This class has similar constraints to the item class. Due to its simplicity, the time complexity is of little concern and the only truly necessary thing is that it is easy to maintain.

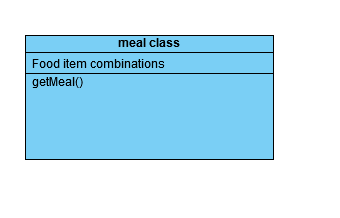
C.4: This class will be storing instances of the item class.

C.5: This class will be storing instances of the item class and acting as a quick and easy way for the front-end developers to spit all of the items into the order summary when the user selects one of the meals on the front end. This class will be used by the order class, the database class, the GUI class, and the ordering system class.

C.6: This class is dependent on the functionality of the item class.

C.7: This class is most likely going to store an ArrayList of Item classes as well as provide getter functionality for retrieving the item classes as the ordering system class and GUI class requires. It will populate that ArrayList uniquely per meal in the constructor.

C.8: This class is a service that is required by every other class in the project except the item class. This class will be simple similar to the item class (they are structured the same way).



#### Figure 6 | Class diagram depicting the Meal class

1. **Order Class**

D.1: This is the highest level of data storage objects. It is going to actively store the items that we add to the order from the front end.

D.2: This class will store all of the items that are being added to the order that is currently being taken. It will be able to pull the item objects out of a meal class as well as provide getter methods that the GUI and database classes will be able to use to extract data from them.

D.3: This class, like the previous two, is extremely simple and thus has little to no constraints.

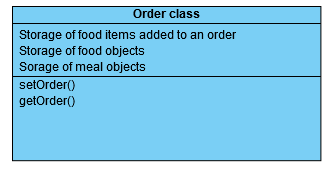
D.4: The subclasses of this class are the item and meal objects. The meal objects simply get extracted to multiple item classes in this class but there will a list of item classes kept inside the order object, making it highly dependent on the item class.

D.5: This class will be heavily used by the ordering system class, the GUI class and the database class as a data object. This class will require expected behavior from the item class and the meal class. This will all be in an effort to maintain OOP within the project.

D.6: This class will require external resources. From within the project it will require the item and meal classes to be functioning properly.

D.7: This class will store an ArrayList of item objects and as items are selected in the GUI the ordering system class will add item objects to that list. If a meal is selected this class will extract the items from that meal object and add those items to the list of items. There will also be thorough getter implementation so that the list of items can be sent to the database and ordering system classes conveniently. Unlike the previous two data storage classes this one will not require hard coding in the constructor as it will be populated on a per order basis.

D.8: This class will be used by the GUI class, the database class and the ordering system class while this class will use the item class and the meal class to fulfill its purpose.



#### Figure 7 | Class diagram depicting the Order class

1. **Database Class**

E.1: This is the class that will upon completion of an order export it to a .txt file and save it in a dedicated directory.

E.2: Essentially this class is responsible for creating a log of the orders so that if customers have complaints about an order being taken wrong there will be a record of exactly what was put into the computer. This class will not provide any services to other classes in the project.

E.3: The only constraint for this class is that it cannot be used to make more .txt files than the drive that the program is being run on can store. It will be the stakeholders responsibility to ensure that this is never a problem, otherwise errors will begin to occur throughout the program.

E.4: This class will make use of pretty much every other class in the project but it will not keep any of those classes instantiated locally within it. Rather, it will be constructed with an order object and executed with a method inside the class to perform the backup.

E.5: This class will make use of all the other classes in the project with the exception of the GUI class but none of the other classes in the project will make use of this class.

E.6: This class will require access to the Date/Time library in Java as well as the ability to make files in the database directory. In terms of internal resources this class will make use of every single data storage object.

E.7: The way this class will work is that it will get constructed with an order object. It will then create a file and use the data in the order object to print the order out to that .txt file. It will also include information such as the date and time in that file. These files will all be printed out to the same directory where the customer can use Microsoft Windows 10’s built in sorting tools to sift through the database.

E.8: This class provides no services to the rest of the project but it requires the services of all the data storage objects to print the data and the services of the ordering system class to trigger it.

1. **Enum for sauces/ingredientless items**

F.1: This is a java enumeration that will be used to store items that do not have ingredients that can be removed, such as sauces.

F.2: This is primarily a data storage class that will have some basic getter functionality to allow the ordering system class and GUI class to pull all the items it stores at the same time. We could just use item classes to store these items but given the items that are being stored by this enum are much simpler than something like a burger, it made sense to use a more simple data structure.

F.3: There are no constraints on this class other than that during maintenance the team maintains the formatting established in the initial development to avoid problems.

F.4: This is a raw java class. It does not have any subcomponents or classes. It is structured as a Java enumeration. Any subclasses of that structure will have to be found in Java’s documentation.

F.5: This class will be used by the GUI and ordering system class to quickly populate all items that do not have ingredients that can be changed out. There is no reason to store these items in a class because they are cost free to add to the order and have no replaceable ingredients making them up that need to be stored. This class will not use any other classes in the project.

F.6: This entity will not require any external resources, nor will it require internal resources.

F.7: This is an extremely basic Java enumeration. Documentation from Java can be found to explain how these work if it is not already known to the reader. There will also most likely be a getter that will spit all the values in the enum back out to the caller in a string array.

F.8: This class will provide services to the ordering system and the GUI class. It will not require the services of any of the other classes in the project.

1. **Ordering System**

G.1: This class will be the direct backend of the GUI class. The GUI class will make calls to it and it will manage the data structures and flow of the ordering process.

G.2: This component is in essence the backbone of the backend of the program. It will manage the flow of the ordering process as well as maintaining the data structures as the order is progressed through.

G.3: This class will have no external dependencies but it is reliant on the functionality of everything except for the GUI class to function correctly.

G.4: The classes used by this class will be everything except for the GUI class.

G.5: This class will make use of everything except the GUI class and the GUI class will make use of ONLY this class.

G.6: There are no external resources required for this class to function. Only the internal ones that were already covered.

G.7: This class will work by providing methods that the GUI can call when buttons are pressed. There will be X major functions: getMeal, getItem, modifyItem, deleteItem, finish and getOrder. getMeal will add the selected meal from the GUI to the data structure, getItem will behave similarly. modifyItem will allow us to make changes to an item already in the order (by making setting the boolean ingredients to true or false depending on what the customer wants). deleteItem will remove an item from the data structure and finish will trigger the database class and end the ordering process. After ending the order things will be reset back to the way they were but the log of the order will remain in the database.

G.8: This class will essentially be an API to the data structure of an order that the GUI class can make use of. As far as classes this class will use - it will pretty much use every other class except for the GUI class.

#### 

## Other Design Features

There are no other intended design features for this project but this section will remain available in the likely event that changes are made during the development of this project.

## Requirements Traceability Matrix

Provide reference to the location of the Requirements Traceability Matrix that indicates traceability from the system requirements documented in the System Requirements Specification to the design elements documented in the System Design Description.

### Table 2 | Functional Requirements -> Classes

| Functional Requirements | Classes |
| --- | --- |
| FR1 | GUI Class |
| FR2 | FoodItem Class |
| FR3 | Meal Class |
| FR4 | Order Class |
| FR5 | DatabaseOutput Class |
| FR6 | Ingredient Class |
| FR7 | IngredientlessItems Enum |
| FR8 | GUI API Class |



College of Sciences & Liberal Arts  
Department of Computer Science

**CS 471 Software Engineering**

**Winter 2022**

**Course Project**

# Software Test Plan

For

Restaurant Ordering System

Version 1.0

CS 471 - G3

Dr. Jamal Alhiyafi

2/28/2022

**Table of Contents**

[Software Test Plan](#_ozjy3eb5db01) **1**

[1. Introduction](#_1y8khzwk73vf) 4

[1.1 Objectives](#_uxvru0b1n1e3) 4

[1.2 Testing Strategy](#_yar477x1xo03) 4

[1.3 Scope](#_5uff455bde4v) 4

[1.4 Reference Material](#_185j391twe7q) 4

[1.5 Definitions and Acronyms 5](#_nof51xu0rcni)

[2. Test Items](#_1ame68kdh1p6) 6

[2.1 Program Modules](#_hzqz1ujzfqvy) 6

[2.2 Operator Procedures](#_w9t8imrcvlxl) 6

[3. Features to Be Tested](#_632qvys6ygap) 7

[4. Features Not to Be Tested](#_e41c7k55g037) 9

[5. Approach](#_vxtcjzmbb10b) 10

[5.1 Component Testing](#_qz8yrci3wsf8) 10

[5.2 Job Stream Testing](#_idouo47vkboc) 10

[5.3 Performance Testing](#_v2p3iiekev4y) 10

[5.4 Regression Testing](#_ngghrx25pwz5) 10

[5.5 Acceptance Testing](#_14ty141g9kkf) 10

[6. Pass / Fail Criteria](#_h7tlndr9807g) 11

[6.1 Suspension Criteria](#_8c4rvsymgsck) 11

[6.2 Resumption Criteria](#_ucsrwxu8ddfc) 11

[6.3 Approval Criteria](#_4oguarj0oies) 11

[7. Testing Process](#_n3ccy61w7kmo) 12

[7.1 Test Deliverables](#_rd6oxg6rfzra) 12

[7.2 Testing Tasks/Cases](#_lzwmalz43lol) 12

[7.3 Responsibilities](#_mkcv6eaia9hu) 13

[7.4 Resources](#_h1nsyuqnwu5y) 13

[7.5 Schedule](#_s0xjyjfkicew) 13

[8. Environmental Requirements](#_10fsufnxwelv) 14

[8.1 Hardware](#_b4ddnq6lkltz) 14

[8.2 Software](#_oqxxcj3adn6p) 14

[8.3 Security](#_n2m853brfpaf) 14

[8.4 Tools](#_huyz8d431xcr) 14

[8.5 Publications](#_r7re1a2m9y6) 14

[8.6 Risks and Assumptions](#_cas6kgmvwil) 14

## Revision History

| **Name** | **Date** | **Reason For Changes** | **Version** |
| --- | --- | --- | --- |
| Cameron Manning | 3/8/22 | Initial fill out | 1.0 |
| Ethan Knott | 3/8/22 | Initial fill out | 1.0 |
| Ethan Knott | 3/16/22 | Making changes requested by professor | 2.0 |
| Owen Bickel | 3/18/22 | Grammar and formatting. | 2.0 |

## 1. Introduction

### 1.1 Objectives

Testing will be done when builds are complete to ensure that bugs are fixed when they are discovered and before it leads to a major issue. There will also be a testing phase that is done when the project is thought to be complete. The objectives of the test plan are to ensure that the program works as designed and without major flaws. The tests will cover a wide range of actions that the program is capable of, as well as testing actions that could appear during normal use but would not be expected like clearing the orders if there are no orders, making a meal a meal, etc. After testing is complete the testing cases and the tasks along with the results will be a deliverable.

### 1.2 Testing Strategy

The test cases listed in section 6 of this document will be run once the program has been fully developed. There are tests that can be run as each level of development is completed. Features that have already been tested previously will still need to be tested as features are being added to ensure no conflicts exist. It is the role of the developers to do testing while the program is being created and making sure conflicts do not exist. It is the responsibility of the testers to ensure that the test cases have been passed and to make sure that the program acts as expected and to report any issues to the developers immediately for remediation.

### 1.3 Scope

Testing will be performed at several points in the life cycle as the product is constructed. Testing is a very 'dependent' activity. As a result, test planning is a continuing activity performed throughout the system development life cycle. Test plans must be developed for each level of product testing.

### 1.4 Reference Material

* Manning, C., Knott, E., Xia, D., Bickel, O. and Hayward, T., 2022. *Project Proposal*. Undergraduate. Kettering University.
* Manning, C., Knott, E., Xia, D., Bickel, O. and Hayward, T., 2022. *Group 3 SDS*. Undergraduate. Kettering University
* Manning, C., Knott, E., Xia, D., Bickel, O. and Hayward, T., 2022. *Group 3 SPMP*. Undergraduate. Kettering University.
* Manning, C., Knott, E., Xia, D., Bickel, O. and Hayward, T., 2022 *Group 3 SRS*. Undergraduate Kettering University.

### 

### 

### 1.5 Definitions and Acronyms

* SDS: Software Design Specification
* SPMP: Software Project Management Plan
* SRS: Software Requirements Specification

## 2. Test Items

### 2.1 Program Modules

All of the modules shall be tested by at least the developer and one other team member. All functions shall be tested thoroughly to ensure that all the functionalities work properly within the parameters. After the developer does the initial functional test, they should alert someone else and have them also test.

### 2.2 Operator Procedures

Java will be required for the program to run. To ensure the program runs in a production environment, the program will be run in a fresh windows 10 environment with java installed to make sure that no libraries (if applicable) are missing. If there is an issue with the program running, reach out to the help desk by phone or email and a technician will get back with you within 30 minutes.

Help Desk procedures: Check to make sure that java is installed. If java is installed, attempt to update java and if that does not work uninstall java from the machine and then reinstall java. If that does not fix it, reach out to Cameron Manning, the project manager.

## 3. Features to Be Tested

* **GUI Class**
  + ○ The Gui class will be tested as each function is implemented into the GUI. When new parts of the program are added to the GUI, it will require a full retest of the system to ensure that everything is working properly.
* **Item Class**
  + This class will be simple to test as it is a simple object containing mostly getters and setters. That said, it also contains an array of Ingredient objects that the class also manages. The only method in this class that is complex enough to justify testing is the removeIngredient method as it involves locating an ingredient inside of the list and removing it from that list. The best way to test this method will be to provide it with a dummy list, use the method to remove an item from that list and then simply check if the item is still there or not. To test this graphically we will know that there is potentially an issue with this class if ingredient lists are not loading properly or the Order data structure is not functioning properly.
* **Meal Class**
  + Similar to the Item class, this is simply an object class. It stores a list of food items that can be retrieved to be added to the list of items in an Order object. The functionality of this object is trivial as it is composed entirely of getters and setters and the functionality of the list is handled by the Java library ArrayList. On the off chance something happens to be wrong with this object it can be graphically detected by GUI elements that populate via the Order data structure not functioning correctly. However, it is much more likely that this class is being misused than that it has any problems. As far as the testing method, this method only ought to be tested at a component level - it is too trivial for unit tests.
* **Order Class**
  + The Order object is the most technically complex class in the backend. It reaches into the Item Class as well as the Ingredient Class that the Item Classes contain. These complex methods are present as this class is where the Ordering System Class (which behaves as an API) directs most of its calls. While the Order Class does contain trivial methods it does have several methods that embed deep into other classes. Those methods are deleteFoodByID, getItemNameByID, getIngredientsByFoodID, addIngredientByFoodID and removeIngredientByFoodIDAndIngredientName. To clear up any confusion, the FoodID is the unique number that each item receives upon instantiation. This is to track which food item the user may be removing in the event of a duplicate item being in the order. All of the previously mentioned methods can be tested by creating a dummy Order (that is populated with food items and ingredients) and simply performing the methods on that dummy Order and checking to see if the desired outcome occurred. The methods do what they say in their name so there should not be any confusion there. Testing this method on a component level is slightly more difficult than the previous two classes. The way one can tell something is wrong with this object is the same as the previous two classes though. The only difference is that a developer ought to check THIS class for bugs BEFORE checking the previous two as this one is significantly more complex than the preceding objects.
* **Database Class**
  + This class has yet to be developed so in depth analysis of method by method testing is not yet possible, but there are still many things that can be assumed on a component level. Any issues regarding the opening of the text files or issues finding an appropriate directory on the host machine will result in a program crash. If the program is not crashing then chances are the program is either not attempting to open the file or the file is being opened without issue. Another potential issue would be unexpected output in the resulting txt files or improper formatting in those files. Any bugs with that sort of problem are most likely due to this class. Interestingly, this class can also be used to detect issues with the Order data structure. If there are problems within the Order data structure then this class will print exactly what is stored in that structure out. If that output is not what was expected then we can derive that there are issues in the Order data structure.
* **Ingredientless Items Enum**
  + This class is a trivial Java enumeration class. It essentially functions as a convenient way to store items that do not contain “ingredients” (think of things like Ranch Dressing and Ketchup). It contains no complex methods and requires no unit tests. On a component level we will know if something is wrong with this class because items like sauces will be behaving in ways we are not expecting while all other items will not be having problems.
* **Ordering System Class**
  + This class is where all of the magic happens. It is the most surface level class in the backend and it offers an API that GUI team can use to create the front end of the ordering system. It contains the order object, allows for manipulation of every part of every object in the Order data structure and will also include triggers for the Database Class to back up a finalized order. Almost all of this class's complex methods are simply calls to the Order class so they are trivial, but it does contain a method called getTotalPrice that is not. This method tallies the total price of all of the items in the current order. The best way to unit test this is to simply create an order with a few items of different prices and check that the result of the method is correct. Graphically any issues will become apparent when there is a disconnect between the total price of the items in an order and the total price being displayed below the order.

## 4. Features Not to Be Tested

Some features will not be tested: we will not be testing on operating systems other than Windows 10 release 21H2, Java versions under 17 and the input will be tested on a laptop, both through touchscreen and mouse clicks. The program will also not be tested in high wind conditions. We also did not test the program on a resolution besides 1920x1080.

## 5. Approach

### 5.1 Component Testing

The main program is the GUI program which will allow for the other components of the system to be linked together. The GUI program will be the first one tested to ensure that the buttons work properly and the screens operate as expected. Next, a part of the program will be tied into the GUI and then tested to ensure that the GUI is working as expected with the new functionality implemented. This process will repeat until all portions of the program are implemented into the GUI class and then the program can be tested as a whole.

### 5.2 Job Stream Testing

The environment needs java installed to operate correctly. The program will not run if java is not installed.

### 5.3 Performance Testing

We will be testing the performance by ensuring that each action on the GUI gives some response within 0.5 seconds. We will also test its availability by leaving it running on a test computer for an extended period of time to make sure that it can handle running for an entire work day.

### 5.4 Regression Testing

We will test all of the test cases once again after all of the code is finished. This will ensure that nothing that was developed at the end affected anything that was already written.

### 5.5 Acceptance Testing

In order to ensure that the software will work on a customer’s machine, we will run it on a machine that is similar in specifications and will monitor the resource utilization while doing so. We will also ensure that all of our test cases pass prior to accepting it

## 6. Pass / Fail Criteria

### 6.1 Suspension Criteria

Suspension will happen if a test fails or causes other components in the system to fail. Suspension might not be necessary depending on the nature of the component failure and how difficult it will be to fix.

### 6.2 Resumption Criteria

Resumption will happen once the failed test or resulting failed components have been fixed.

### 6.3 Approval Criteria

To approve a test result a few things must happen. When the test is run the actual output needs to match the expected output. The result cannot cause any other issues to components in the system.

## 7. Testing Process

### 7.1 Test Deliverables

After testing has been completed, the test case table will be used as a deliverable. This will allow the stakeholders to see what errors the program has. It will also act as a checklist for the development team to make changes if applicable.

### 7.2 Testing Tasks/Cases

| **Test Case** | **Functionality** | **Input** | **Expected Output** | **Actual Output** | **Pass / Fail** |
| --- | --- | --- | --- | --- | --- |
| Editing condiments on food | Be able to manage condiments that come with a food item | Any main course item, then add or remove a condiment | The condiments on the food item will reflect the changes made | Condiment changes were reflected in the ordering system and the system output | pass |
| Adding a side to main course item | Allows for a side to be added to a main course item using a single button | Any food item and then pushing the add side button | The side should be attached to the main course item | A la carte item was added successfully | pass |
| Remove a side from a meal | Allows for a meal to be downgraded to its food item and drink | Any meal and then removing the side from the summary | You should be able to remove a side from a meal without deleting everything in the meal | Side was removed from the meal | pass |
| Remove all condiments from a food item | When all condiments are removed from the food item, the program should know that the food item is plain | Insert a food item and then edit it and remove all toppings | The program should know the food item is plain instead of having a blank list of condiments | Condiment was removed | pass |
| Add two of the same food item but with different toppings | Each object should be independent even if they are of the same type | Add an item with no toppings and then add the same item with 2 toppings | There should be two different food items listed in the summary | Each food item had correct toppings associated | pass |
| Clicking finish with nothing | Clicking finish with no food items should not make a blank text file | Nothing | The ordering system should not output a blank order file if nothing was ordered | nothing | pass |
| Price adds up correctly | The price should be correct even if items are removed | 2 cheeseburger meals minus one of the drinks | $14.65 | $14.65 | pass |

### 7.3 Responsibilities

We will have a small team of testers who will mainly be ones responsible for managing, designing, preparing, and executing the testing. They will work closely with the developers when preparing the test cases as well as when communicating the output if there are any fail cases. In addition, since the group is so small, the developers will also be responsible for performing and designing some of the tests. This is done not only to split the work of testing, but also to ensure that code gets thoroughly tested and reviewed by more than just two members.

### 7.4 Resources

The development team will do testing while the program is being created and ensure the different modules work together as expected. The testing team will be responsible for doing the test cases and marking if they passed or failed.

### 7.5 Schedule

The gui class will be the first class developed and tested. The gui class will make calls out to other classes that will then return information back to the gui to be displayed to the user. Each part of the program will be tested with the gui and as more classes are added, testing will reveal if there are conflicts between classes. Testing each class with the gui class should take around 5 minutes and with each additional class being added, it should add an additional 3 minutes to test. Testing the test cases at the end should take 10 minutes

## 8. Environmental Requirements

### 8.1 Hardware

The only pertinent requirement for the testing of our software is access to a Windows 10 capable computer. Networking hardware is not necessary as our software does not make use of an internet connection (locally or otherwise).

### 8.2 Software

Our software does require that the host computer is running Windows 10, has enough storage (ideally several gigabytes of storage) to store as many order backups as the client wants and has the latest version of Java installed. Those are the requirements to simply use the program. To properly test the program a developer will have to have the JDK installed as well as an IDE to efficiently manipulate the code.

### 8.3 Security

Given this software has no critical data (user or otherwise) that it manipulates there is no need for any elaborate security measures. The only concern is that the product works, not that it is secure (for the previously mentioned reasons). The only security concern that the developers are taking note of is using updated and well maintained libraries to ensure that none of the libraries used to create the software are open to things like RCE attacks (Remote Code Execution). All other security is the responsibility of the customer.

### 8.4 Tools

No external tools will be used for testing. Testing will be conducted during the development and after development when the program is ready to be checked against the test cases.

### 8.5 Publications

The two most important documents for supporting testing activities are this document and the SRS. The SPMP may also be useful in the event the reader wants to contact the developer responsible for a certain section of code.

### 8.6 Risks and Assumptions

This development group has a tendency to get a lot of work done close to deadlines. Testing often falls to the wayside in development environments like this. To counteract this we have dedicated two team members to code documentation and testing exclusively. Ideally this will encourage a good deal of effort to be put into tests. The other concern actually stems from the fact that two team members are focusing on testing and documentation rather than coding. Due to the current team's structure there is an expectation that the testing team will handle testing in its entirety - the people doing the initial development are not even going to touch that process. This makes it quite likely that the testers will end up confused on how to properly test something or what a certain line of code is doing. The best way to handle this is for the initial developers to simply be reachable as much as possible. We are using Discord to communicate which gives all of us near 24/7 access to one another in the event someone quickly needs an answer to a question.

#### Lessons and Skills Learned

Throughout the completion of this project we learned quite a bit as a group. When we started out we were relatively disorganized and confused on what was and was not expected of us. As the term progressed we gradually began to learn a bit more about each other and what the general expectations were and the quality of our work improved. Overall it was a good term and every member of the group was able to experience a healthy amount of growth.

As far as the lessons we learned, one of the most important things that we learned was that waiting until the last minute to deal with group work is a nightmare. We had an instance where we had all been working on a paper all the way up to the last minute and nobody thought to actually turn the thing in. If it wasn’t for the fact that someone remembered that it had not been submitted ten minutes before the deadline then the work would have been turned in late. While we were never able to get organized enough to the point that this behavior stopped occurring it is reasonable to say that we all learned that dealing with this sort of stuff hanging over our heads is not a fun way to live. Another lesson that was learned was that allowing too loose of a leash results in problems like the ones mentioned above. If a group leader leaves group members to their own devices and defaults to the professors due dates rather than setting up an expectation for work done prior to that due date then there will always be a mad rush to get everything done the day the assignment is due. One thing to take note of is that every time the group leader actually created one of these schedules the work would be done ahead of time and the quality of the work would reflect that. A lot of leaders fear being annoying but perhaps taking a bit more control of a situation and making up for the annoyance that causes in other ways is a better approach to leading a small group.

The group learned a great deal of skills that will be heavily applicable in the field. The way our project was developed required an API to be connected to a GUI, a design pattern that is seen everywhere in Computer Science. The API developer was able to experience developing a backend without completely understanding how the implementation of the front end was going to work. This resulted in some methods that front end development needed not being present as well as some unnecessary methods being present in the API. The group held a meeting to connect the two ends of the project and after four hours of work we had successfully worked through about a dozen problems and had a working prototype. This experience was extremely valuable as we got to work as a team on actual code and bring together a project that was worked on by multiple people. While these skills were not as applicable to the actual project the group leader got a great deal of experience learning what it is like managing developers and what all goes into that process. Even if a student does not end up pursuing management as a career path it is still valuable to have an understanding of what goes into managing developers properly. Another useful skill that everybody in the group learned was how valuable good planning is. Some of the stuff we planned on doing was outside the scope of what was possible for our group with the time we had. We ended up having to adjust several of the planning documents to reflect what we were actually able to achieve. Due to this confusion, things like the API ended up having extra functionality that was unneeded. Proper planning and documentation could have potentially prevented that from being a problem, but luckily it was not that big of a deal in this case. That said, it’s good to learn that lesson now rather than when large sums of money and careers are on the line.

#### Conclusion

At the beginning of the term, we set out to create a cost-effective and efficient order taking system for small restaurant businesses. To accomplish this, we kept the user interface and the backend systems as simple as possible. This would help keep the application lightweight while also being easy for a new user to become familiar with quickly. The end result was exactly what we had envisioned in terms of functionality and useability. With an easy to use interface, users are able to quickly add any item from the menu to the order and can customize the food items to the customer’s liking. Once the order is done, a single click will finalize the order and send it to the database which is easy to access if there is any need. In addition, the program uses very little resources to accomplish the goals we set out to accomplish. We were able to implement all of our functional requirements while still keeping the program simple to understand and us

#### Appendix

**Appendix A: Presentation Slides**

The presentation slides will be provided at a later date via a separate submission.

**Appendix B: Program**

The software that this document discusses will be included in aseparate file in the project folder

**Appendix C: README**

The readme file will be provided in a separate file in the project folder.