**Software Requirements Specification**

**for**

Lower Wismer Ordering Service

**Version 1.0 approved**

**Prepared by Matthew Parias, Amanda Laucks, Samuel Gregory, Thomas Rohrbach**

**The Wismer Weebs**

**11/04/20**

**Table of Contents**

**Table of Contents ii**

**Revision History ii**

**1. Introduction 1**

1.1 Purpose 1

1.2 Document Conventions 1

1.3 Intended Audience and Reading Suggestions 1

1.4 Product Scope 1

1.5 References 1

**2. Overall Description 2**

2.1 Product Perspective 2

2.2 Product Functions 2

2.3 User Classes and Characteristics 2

2.4 Operating Environment 2

2.5 Design and Implementation Constraints 2

2.6 User Documentation 2

2.7 Assumptions and Dependencies 3

**3. External Interface Requirements 3**

3.1 User Interfaces 3

3.2 Hardware Interfaces 3

3.3 Software Interfaces 3

3.4 Communications Interfaces 3

**4. System Features 4**

4.1 System Feature 1 4

4.2 System Feature 2 (and so on) 4

**5. Other Nonfunctional Requirements 4**

5.1 Performance Requirements 4

5.2 Safety Requirements 5

5.3 Security Requirements 5

5.4 Software Quality Attributes 5

5.5 Business Rules 5

**6. Other Requirements 5**

**Appendix A: Glossary 5**

**Appendix B: Analysis Models 5**

**Appendix C: To Be Determined List 6**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
|  |  |  |  |
|  |  |  |  |

# **Introduction**

## **Purpose**

This document describes a phone app which allows a student to order food from lower wismer. It will send a request to the restaurant which is being ordered from. The workers will then make the food and send out a push notification to the student who ordered. All food is paid for when the food has been made and the notification is sent. This SRS will describe the entire system of how the customer/student will interact with the program, how the employees who make the food will interact with the program, what happens behind the scenes, and how any admin could change aspects of this program. This is the first release of this program.

## **Document Conventions**

We decided to use Arial font, size 11.

## **Intended Audience and Reading Suggestions**

ThIs document is intended for Veca Schilling, the CS275 professor, and project managers and developers for technology at Ursinus that would receive this product pitch if applicable. It is recommended that the reader work through each section of this document in numerical order.

## **Product Scope**

People on campus who order food from lower Wismer can make use of the app. People who use upper wismer/ C-Store wouldn't be able to take advantage of the app. People who don't go to Ursinus also won't be able to benefit from the app as it isn't for their school. It is useful for customers to pick up their food when it’s ready. Useful in the sense of following covid guidelines, minimizing possible infection rates, and spreading at school.

## 1.5 References

*This document follows Ursinus’s CS style guide for coding which can be found* [*here*](http://www.ctralie.com/Teaching/CS174_F2020/style.html)*.*

*“CS 173/174 Style Guide.” Ursinus CS 174: Object Oriented Programming, Fall 2020, www.ctralie.com/Teaching/CS174\_F2020/style.html.*

# **Overall Description**

## **Product Perspective**

Lower Wismer Ordering Service is a program made for the students of ursinus to replace the convention of ordering food on a device within lower wismer dining. This program is being built for the purpose of a final project for CS275 at Ursinus college. The application will be used on android/IOS and be able to function as a temporary application until we are able to implement it in to the existing Ursinus “MobileU” app.

## **Product Functions**

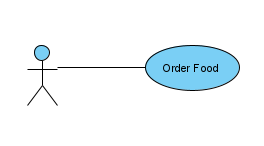
* The product must present to the user a Graphical User Interface (GUI) upon start-up which will prompt the user to sign in, after. It will have five buttons: Tres Habenaros, Sub-Connection, The Grill, and Jazzman’s and saved orders
  + If Jazzman’s is closed due to time of day, this option will not appear.
* The user must select one of the five buttons.
* Upon selection, it will open up to many buttons in a list that the user can scroll through. Each button will represent a menu item from each of the respective restaurants, as well as one to go back to the main menu.
* Once order is complete you can either add another item or finish your order and pay.

This section will describe the use cases for each the Student, WIsmer worker, and the developer.

**2.2.1 Student use case**

Use case: **Order Food**

**Diagram:**



**Brief Description**

The student starts their order and accesses different restaurants from lower.

**Initial Step-By-Step Description**

Before this use case can be initiated, the student must have opened the app

1. Student signs in to his account

2. Student selected restaurant

3. Student picks main order

4. Student selects if he wants toppings

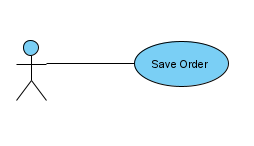
5. Student selects if they want a combo with there order

6. Student selects to complete his order or add another item.

**2.2.2 Student use case**

Use case: **Save Order**

**Diagram:**



**Brief Description**

The student Requests his order to save within the app where they’re able to see updates on his

order / present his order to Wismer worker

**Initial Step-By-Step Description**

Before this use case can be initiated, the student must have opened the app and ordered

1. selects to save his order

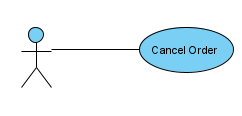
2. Student selects yes to confirm saved order

3. Student can select to save his order to the phone

**2.2.3 Student use case**

Use case: **Cancel Order**

**Diagram:**



**Brief Description**

The student decides to cancel his order either before it is accepted by the wismer worker or

before he finishes his order.

**Initial Step-By-Step Description**

Before this use case can be initiated, the student must have opened the app and in the process of ordering.

food. If already ordered, go to step two.

1. Student selects order to cancel from his order screen before it is place

2. Go to saved order screen

3. Student selects the order he wants to cancel

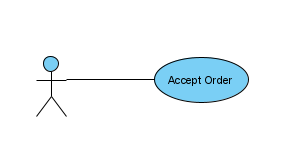
4. Student selects cancel on order

5. Student selects to confirm if they want to cancel the order

**2.2.4 Worker use case**

Use case: **Accept Order**

**Diagram:**



**Brief Description**

The worker will select an order to accept on his screen which will begin order making

**Initial Step-By-Step Description**

Before this use case can be initiated, the student must have placed an order

1. Worker receives order

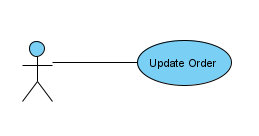
2. Worker will accept an order from there screen in lower wismer

3. Worker confirms order start

**2.2.5 Worker use case**

Use case: **Update Order**

**Diagram:**



**Brief Description**

The worker will send updates of order to database for/completion of order

**Initial Step-By-Step Description**

Before this use case can be initiated, the student must have placed an order and the order had to have

accepted an order.

1. Worker selects specific order

2. Worker selects to either update time or completion update

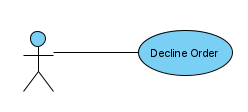
3. Worker selects to confirm time / completion

4. Worker send update / notification

**2.2.6 Worker use case**

Use case: **Decline Order**

**Diagram:**



**Brief Description**

The Worker decides to decline an order if too many orders are up or if ordered food isn't in stock.

**Initial Step-By-Step Description**

Before this use case can be initiated, the student must have opened the app and ordered something out of stock

1. Worker receives order

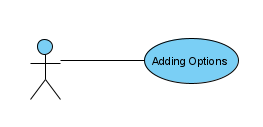
2. Worker declines order

3. Worker confirms to decline order which sends notification to Student

**2.2.7 Developer use case**

Use case: **Adding Options**

**Diagram:**



**Brief Description**

The Developer is able to access the database to add a new restaurant food option.

**Initial Step-By-Step Description**

1. Developer signs in to database

2. Edits database with new options

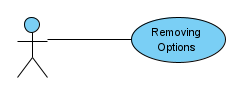
3. Developer tests the adding of new items

4. Developer saves the changes to database

**2.2.8 Developer use case**

Use case: **Removing Options**

**Diagram:**



**Brief Description**

The Developer is able to access the database to remove a restaurant food option. This may be due to it being

out of stock or being discontinued.

**Initial Step-By-Step Description**

1. Developer signs in to database

2. Edits database by removing options options

3. Developer tests the system after removing items

4. Developer saves the changes to database

## **User Classes and Characteristics**

The highest user class is the super admin which will be available to our developers. There should also be a user class for those further developing the app in the future. There has to be a user class for tech support to maintain the app. The workers will need a user class to be able to view orders for their respective restaurants. The person ordering will have a user class and will just be able to use the app.

The Student will need to be able to download and interact with either “Lower wismer ordering service” or “MobileU”. The student also needs to use simple button functionality within an application. The product needs to satisfy the student sufficiently as an alternate way to order.

The Developer will need to be able to access the database and know the basics of a database to operate and test functionality. He will also need to know how to work a basic phone application.

The Wismer worker will need to be able to interact with his screen of orders and know how to accept and change orders from his end.

## **Operating Environment**

Android Studio 4.1.1 is the official Integrated Development Environment (IDE) for Android app development and will be used as software. Hardware platforms used will be laptops and cell phones.

## **Design and Implementation Constraints (Optional)**

We will have to implement the devices used by the Lower Wismer workers for the application to be compatible. We are limited in the coding languages for developing a mobile application to koltine, C++, and Java. As software to use as a database we will use MySQL and to test the database itself we can use MySQL workbench. As stated before we are also constrained by Ursinus’s style guide.

## **User Documentation**

The app wouldn’t come with extra resources for use. In the app, there will be help icons for the user to navigate and order when they need extra help. Our design should be simple enough for most users to understand.

## **Assumptions and** Dependency

This program assumes that the user is an Ursinus student because the user must log in with their Ursinus username and password in order to confirm the use of dining dollars or meal swipe. A constraint within this project is the ability to access the software used by kitchen staff.

# **External Interface Requirements**

## **User Interfaces**

*<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.>*

The UI in the phone app will start at a home screen with the 4 different restaurants and a login option. There will be a login page allowing the user to use their school account to order food with their information. After selecting the restaurant, there will be the meal options for that place. There will be a back button present on every page except the home screen, and there will be an option to see the current order while making it. On that page, there is the option to remove items from the list, add more, or finalize the order to send it.

## **Hardware Interfaces (Optional)**

*<Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, and communication protocols to be used.>*

## **Software Interfaces**

*<Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.>*

We will need a database for orders and available food items. We will need our orders to be recognized by the existing ordering interface the restaurants use. There will be data sent between the client and server. To let the user login with the student information, we have to be able to send the secured data to the Ursinus login information to let the user in.

## **Communications Interfaces**

*<Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.>*

We will be sending order data back and forth between the client and server sides. Login information must be encrypted to keep data safe.

# **System Features**

*<This template illustrates organizing the functional requirements for the product by* ***system features, the major services provided by the product.*** *You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.>*

*[Note- Students of CS275 Will most likely organize by bolded topics]*

## **System Feature 1**

*<Don’t really say “System Feature 1.” State the feature name in just a few words.>*

4.1.1 Description and Priority

*<Provide a short description of the feature and indicate whether it is of High, Medium, or Low priority. You could also include specific priority component ratings, such as benefit, penalty, cost, and risk (each rated on a relative scale from a low of 1 to a high of 9).>*

4.1.2 Stimulus/Response Sequences

*<List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.>*

4.1.3 Functional Requirements

*<Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary. Use “TBD” as a placeholder to indicate when necessary information is not yet available.>*

*<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>*

REQ-1:

REQ-2:

## **System Feature 2 (and so on)**

# **Other Nonfunctional Requirements**

## **Performance Requirements**

*<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.>*

## **Safety Requirements**

*<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product’s design or use. Define any safety certifications that must be satisfied.>*

## **Security Requirements**

*<Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.>*

## **Software Quality Attributes**

*<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.>*

## **Business Rules**

*<List any operating principles about the product, such as which individuals or roles can perform which functions under specific circumstances. These are not functional requirements in themselves, but they may imply certain functional requirements to enforce the rules.>*

# **Other Requirements**

*<Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.>*

**Appendix A: Glossary**

*<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.>*

**Appendix B: Analysis Models**

*<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams*.>

**Appendix C: To Be Determined List**

*<Collect a numbered list of the TBD (to be determined) references that remain in the SRS so they can be tracked to closure.>*