Software Requirements Specification

for

Restaurant Ordering Application

**Version <3.0>**

**Prepared by**

|  |
| --- |
| **Ian Murphy** |
| **Neelakshi Soni** |
| **Nicholas Malamud** |

**Group Name: FFTech**

|  |  |
| --- | --- |
| **Instructor:** | **Asif Imran** |
| **Course:** | **CS 441 Software Engineering** |
| **Date:** | **December 14, 2022** |

**Contents**

|  |
| --- |
| **Table of Contents** |
| 1. INTRODUCTION   * 1. DOCUMENT PURPOSE   2. PRODUCT SCOPE   3. INTENDED AUDIENCE AND DOCUMENT OVERVIEW   4. DEFINITIONS, ACRONYMS AND ABBREVIATIONS   5. DOCUMENT CONVENTIONS   6. REFERENCES AND ACKNOWLEDGESMENTS   2. OVERALL DESCRIPTION  2.1 PRODUCT OVERVIEW  2.2 PRODUCT FUNCTIONALITY  2.3 DESIGN AND IMPLEMENTATION CONSTRAINTS  2.4 ASSUMPTIONS AND DEPENDENCIES  3. SPECIFIC REQUIREMENTS  3.1 EXTERNAL INTERFACE REQUIREMENTS  3.2 FUNCTIONAL REQUIREMENTS  3.3 USE CASE MODEL  4. OTHER NON-FUNCTIONAL REQUIREMENTS  4.1 PERFORMANCE REQUIREMENTS  4.1 SAFETY AND SECURITY REQUIREMENTS  4.1 SOFTWARE QUALITY ATTRIBUTES  5. OTHER REQUIREMENTS  5.1 ACTIVITY MODELLING  5.2 SEQUENCE MODELLING  5.3 DATABASE DIAGRAMS  5.4 CLASS DIAGRAMS  5.5 CHANGE LOG  5.6 SOFTWARE EVOLUTION |

**Revisions**

| **Version** | **Primary Author(s)** | **Description of Version** | **Date Completed** |
| --- | --- | --- | --- |
| Draft Type and Number | Full Name | Information about the revision. This table does not need to be filled in whenever a document is touched, only when the version is being upgraded. | 00/00/00 |
| 1.0 | Neelakshi Soni, Nicholas Malamud, Ian Murphy | * This was the first modification made to the document. * We outlined some of the most important system requirements. * Minor changes across the entire document. | 10/15/22 |
| 1.1 | Ian Murphy | * Added sequence diagrams for when the front end wants to finalize an order, get an up to date menu, and when receipt information is requested. * Added the ERD for the backend database. | 10/28/22 |
| 1.2 | Neelakshi Soni, Ian Murphy | * Added class diagrams for the DBController/OrderController/Item hierarchy/Order hierarchy | 11/02/22 |
| 1.3 | Neelakshi Soni, Nicholas Malamud, Ian Murphy | * Added use case diagrams for the customer. * Including modifying their order, canceling, and finalizing their order. | 11/07/22 |
| 2.0 | Ian Murphy | * Updated the diagrams so that no grids are shown in the background. * Combine the use cases into one single clear diagram. | 11/31/22 |
| 2.1 | Ian Murphy | * Updated some sequence diagrams to include the factory classes. * Updated/Added class diagrams for the Item Factory & Order Factory. * Added more database diagrams showing the structure of each table. * Added before and after for the UI versions. | 12/04/22 |
| 3 | Neelakshi Soni, Nicholas Malamud, Ian Murphy | * Updated diagrams to remove borders. * Added UI v3. * Added Database structure images. * Added Github Repo link. | 12/14/22 |

# Introduction

## Document Purpose

This is version 3.0 of a fast-food self-serving kiosk application. This product will enable customers to create and submit orders for themselves. This is done by having the customer interact with our UI, generated from a backend DB, to produce input and our backend will create and submit the order to the employees in charge of handling orders.

## Product Scope

This product will l be able to create food orders based on the customers input to the UI. The application will provide all necessary widgets to make the ordering process for the customer as easy and intuitive as possible. Making this a quick and easy option to order, helping eliminate long lines inside the store front.

## Intended Audience and Document Overview

This document is for our clients, fast food retailers, and our professor Asif Imran. The rest of this document will contain requirement specifications for users and the system. As well as diagrams showcasing the sequence of operations for certain operations, use cases that the customer will come across when interacting with our system, and as well as how our database tables have been related to one another in the back end.

## Definitions, Acronyms and Abbreviations

BE = Back End

DB = Database

FE = Front End

FFT = FastFoodTech

IEEE = Institute of Electrical and Electronics Engineers

JDBC = Java Database Connection

OC = Order Controller

SRS = Software Requirements Specification

UI = User Interface

UML = Unified Modeling Language

API = Application Programming Interface

## Document Conventions

In general, this document follows the IEEE formatting requirements. Use Arial font size 11, or 12 throughout the document for text. Use italics for comments. Document text should be single spaced and maintain the 1” margins found in this template. For Section and Subsection titles please follow the template.

## References and Acknowledgments

CSUSM Standards for Student Conduct:

[*https://www.csusm.edu/dos/studres/standards\_student\_conduct.html*](https://www.csusm.edu/dos/studres/standards_student_conduct.html)

Github Repository Link:

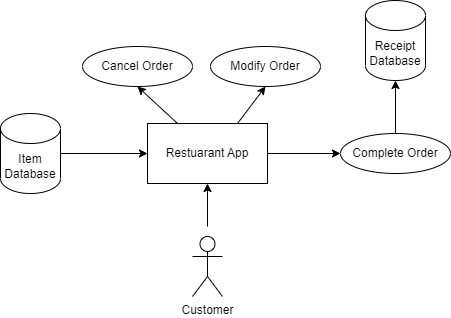
<https://github.com/imurphy92064/FastFood_OrderingApp>

# Overall Description

## Product Overview

Customers will use our application to order food from a restaurant. The application will get info about the items (name and price) from an external item database. The customer can modify the order by adding and removing items. They have an option to cancel the order throughout the process. When they complete the order and purchase the food, their receipt info is uploaded to an external receipt database.

Author: Nicholas Malamud



## Product Functionality

1. User can order any item or items from the menu
2. User can remove an item or items from the order
3. User can see the description of the item
4. User can complete an order
5. User can cancel a in progress order

## Design and Implementation Constraints

1. *We only use Java for this project*
2. *We are using XAMPP that has a mySQL & Apache server to handle BE*
3. *We are also limited on the UI options, currently (10/25) we are using JFrame to create the UI*

## Assumptions and Dependencies

1. Assuming that the customer will create the order over 90% of the time.
2. We rely on MySQL database and WebSocket connection.
3. We rely on the “*WindowsBuilder*” plugin. It's a UI editor to help visualize as you build it.

# Specific Requirements

## External Interface Requirements

### User Interfaces

The interface that the user will interact with will consist of menu items and other buttons that facilitate the order making. There will also be an order cancellation button.

### Hardware Interfaces

*You are not required to specify what protocols you will be using to communicate with the hardware, but it will be usually included in this part as well.*

### Software Interfaces

1. We will be using the WebSocket API that will connect and send messages to the database.
2. The other components will connect to the database controller and issue commands to be sent to the database.

## Functional Requirements

### We shall create a database connection, which can insert, delete, read, and modify tables records.

* + 1. **We shall have a UI that contains all Menu items available and facilitates the ordering process.**
    2. **The system shall create, modify, destroy, and complete orders with input from the customer.**
    3. **The system shall save all orders that have been completed in the database.**
    4. **An order shall have at least 1 item.**

## Use Case Model

### Use Cases

**Author –** Neelakshi Soni / Nicholas Malamud / Ian Murphy

**Purpose** - This diagram shows the activities that the customer is able to do while interacting with the UI. The diagram shows that the user can;

* Add an item to their order
* Remove an item from their order
* Cancel their order
* Finalize and submit the order for preparing.

**Requirements Traceability**

* The App must be able to add and remove menu items from their order.
* The App must be able to cancel a customer’s order.
* The Ap must be able to finalize an order for the customer.

**Priority** - High priority, these are the base functions of our system.

**Preconditions**

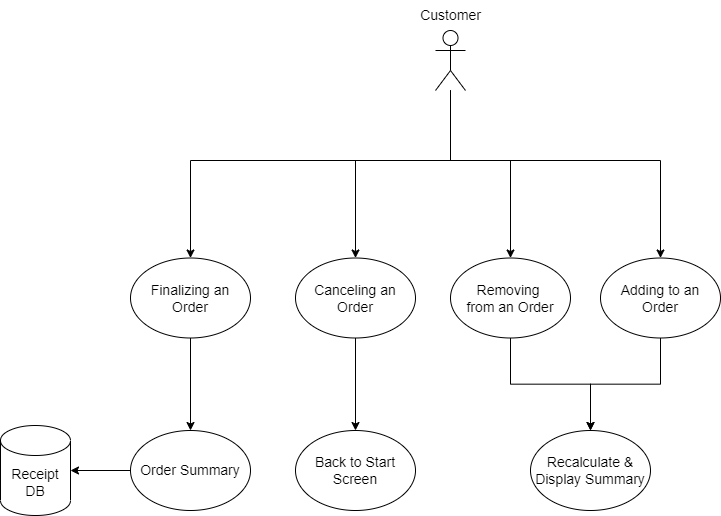
* To cancel an order, an order must already been created
* To remove an item there must be an item in the order already
* To finalize an order, there must be at least 1 item.

**Post conditions**

* After canceling the order all data gathered will be removed for the next customer.
* After removing or adding a menu item, the system will be ready for more input.
* After finalizing the order, the order is converted into a receipt and sent to the DB.

**Actors** – Customer, UIController, OrderController, DBController

**Flow of Events**

* Customers modify their current order, either removing or adding items.
* Customers cancel an in-progress order for whatever reason.
* Alternatively, the customer completes their order and finalizes the order.

# Other Non-functional Requirements

## Performance Requirements

* The product must be operational 99% of the time during in-store business hours of the restaurant.
* There must be tolerable time between the customer giving input to the system and the system processing it.
* There must be tolerable time between the customer giving input to complete an order and the system processing the order and completing it.
* There must be tolerable time between the customer canceling the order and the system resetting the current order for the next customer in line.

## Safety and Security Requirements

The system must be able to save and recall receipts for a certain number of months in order to give customers refunds/fraud protection.

## Software Quality Attributes

For the final version of the product, we want to have a nice, friendly, & inviting UI to encourage the customer to use the product rather than get inline for an in-person interaction to make an order. This may include:

* *Pictures*
* *Color Schemes*
* *Enticing Menu Item Descriptions*
* *UI Adjusts to window size.*

# Other Requirements

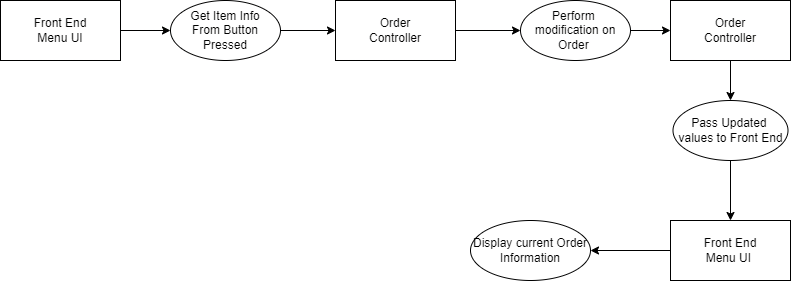
* 1. **Activity Modelling**
     1. **Activity Model #1**

**Author –** Ian Murphy

**Purpose** - This activity diagram shows the flow of events when the FE wants to make a modification to an order being created.

**Requirements Traceability –** The UI will show an up-to-date menu to the customer. (Req. 3.2.3)

**Priority** - This is a high priority.

**Actors** – The FE controller, the OC

* + 1. **Activity Model #2**

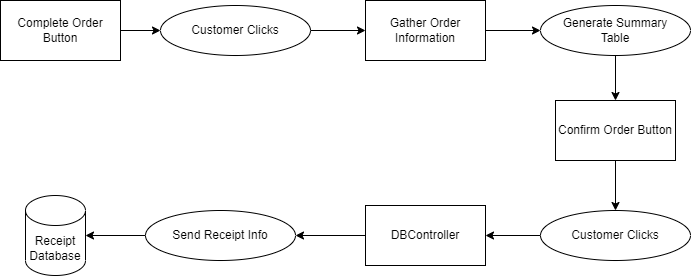
**Author – Nicholas Malamud**

**Purpose** - Complete order and send info to receipt database

**Requirements Traceability –** Need to complete and confirm order and send info to receipt database

**Priority**- High Priority

**Actors** – Customer, DBcontroller, UIController



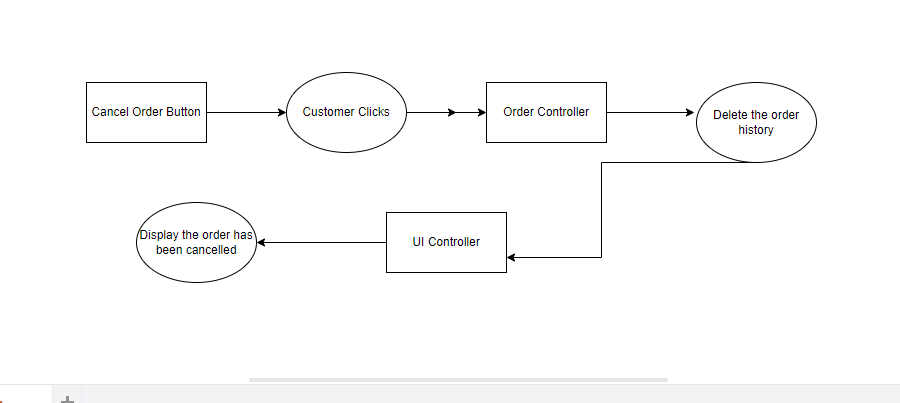
* + 1. **Activity Model #3**

**Author – Neelakshi Soni**

**Purpose** - to cancel the order and display it.

**Requirements Traceability –** Need to cancel order and display it.

**Priority** - High Priority

**Actors** – Customers, UIController and OrderController.

* + 1. **Activity Model #4**

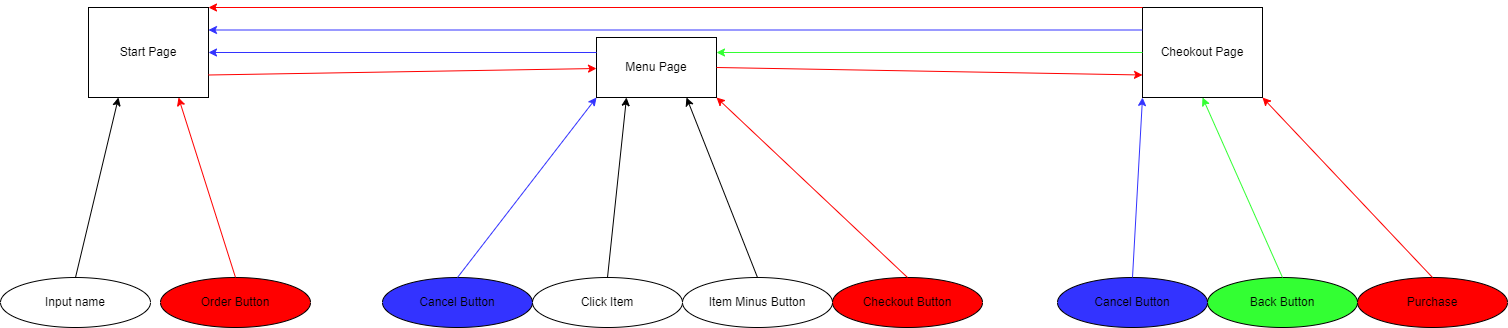
**Author – Nicholas Malamud**

**Purpose** - UI user input and navigation

**Requirements Traceability –** How the UI controller uses buttons to collect user input and navigate through the pages

**Priority**- Medium Priority

**Actors** – Customer, UIController



* 1. **Sequence Modelling**
     1. **Sequence Diagram #1**

**Author –** Ian Murphy

**Purpose** - This sequence diagram shows the sequence of events that will happen when the FE wants to get an up-to-date menu from the BE DB.

**Requirements Traceability –** The UI will show an up-to-date menu to the customer. (Req. 3.2.2)

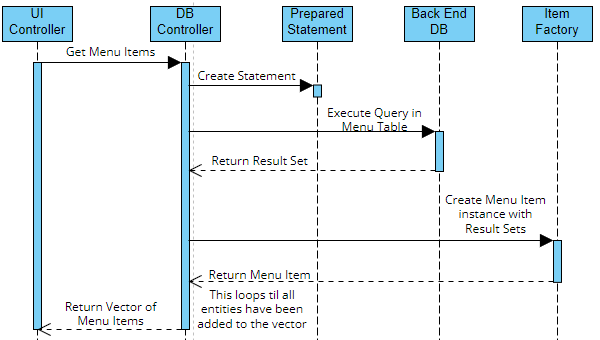
**Priority** - This is a high priority.

**Preconditions** - There must have been a valid and successful DB connection established.

**Actors** – The FE controller, the DB controller, as well as the mySQL server being connected too.

**Flow of Events**

FE controller request Menu -> DB Controller creates/executes query -> DB responds -> Passed Menu items to FE.

* + 1. **Sequence Diagram #2**

**Author –** Ian Murphy

**Purpose** - This sequence diagram shows the sequence of events that will happen when the FE wants to finalize an order the customer has just created

**Requirements Traceability –** The system will allow the customer to complete orders they have created. (Req. 3.2.3)

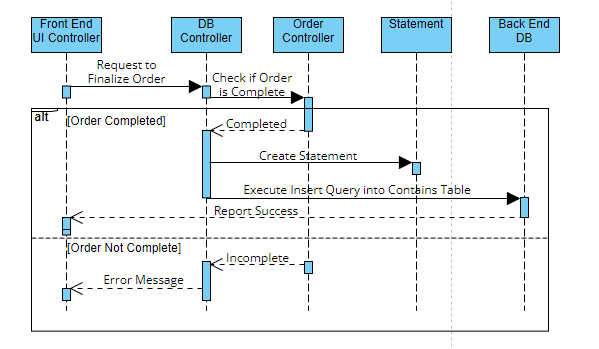
The system will save the order that was completed in the BE DB. (Req. 3.2.4)

**Priority** - This is a high priority.

**Preconditions** - There must have been a valid and successful DB connection established.

There must be a valid order that was completed for this process to finish successfully.

**Actors** – The FE controller, the DB controller, as well as the mySQL server being connected too.

**Flow of Events -** FE controller requests Order Finalization -> DB Controller creates/executes query -> DB adds the entry into the receipt table -> The DB Controller then makes entries in the receipt\_info table to save item quantity info for the order.

* + 1. **Sequence Diagram #3**

**Author –** Ian Murphy

**Purpose** - This sequence diagram shows the sequence of events that will happen when the BE DB needs to insert entries into the receipt\_info table. This table holds item quantity info for receipts that have been added to the receipt table.

**Requirements Traceability –** We shall save the receipts and info of item quantity per receipt (Req. 3.2.4)

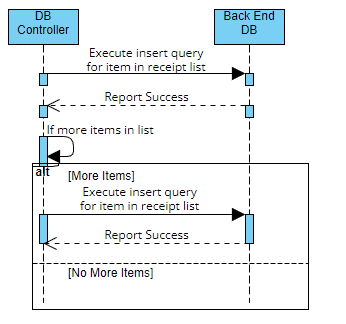
**Priority** - This is a high priority.

**Preconditions** - There must have been a valid and successful DB connection established.

**Actors** – The DB controller, as well as the mySQL server being connected too.

**Extends –** This is an extension sequence from the complete Order sequence.

**Flow of Events**

DB Controller creates/executes query -> DB adds the entry into the receipt table.

* + 1. **Sequence Diagram #4**

**Author –** Ian Murphy

**Purpose** - This sequence diagram shows the sequence of events that will happen when the FE wants to request information about a receipt given a receipt number and item id.

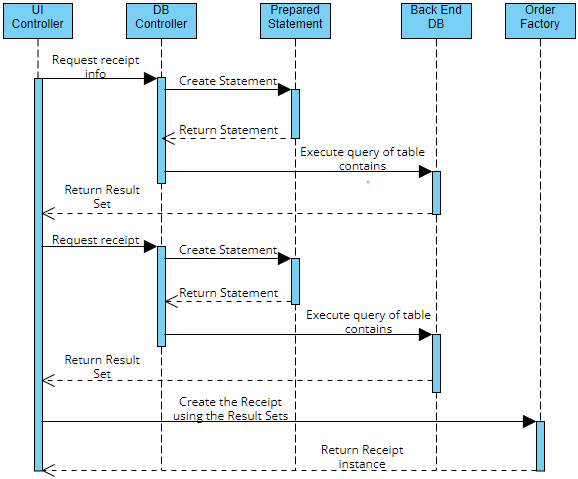
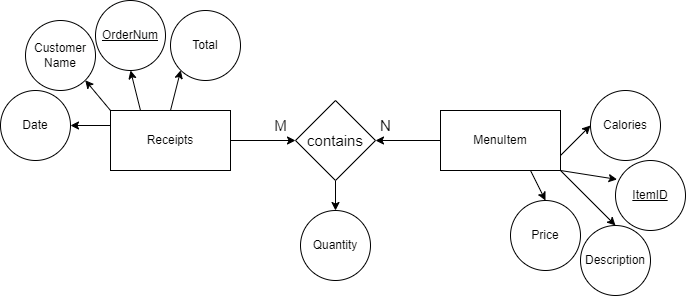
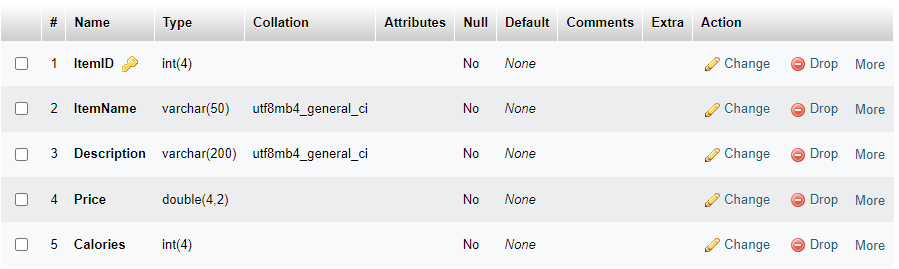
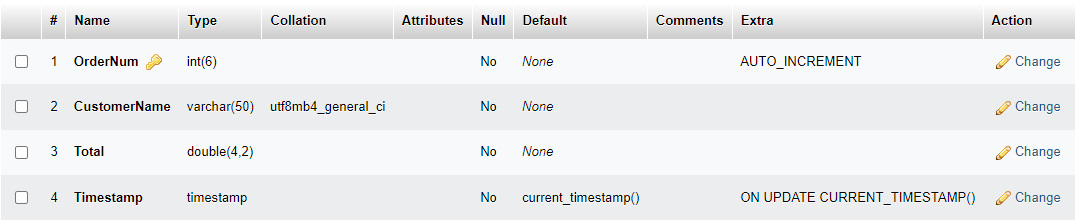
**Requirements Traceability –** The system is able to retrieve information about past receipts (Req. 3.2.4)

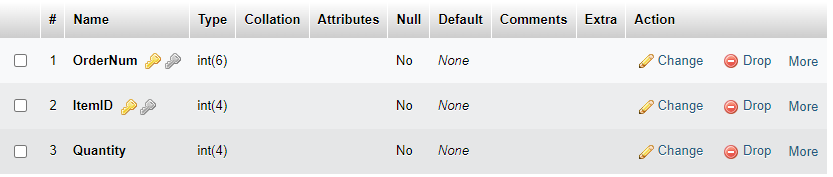
**Priority** - This is a medium priority.

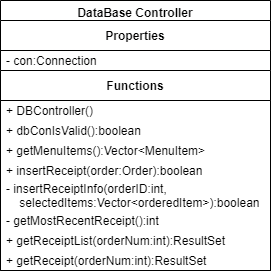
**Preconditions** - There must have been a valid and successful DB connection established.

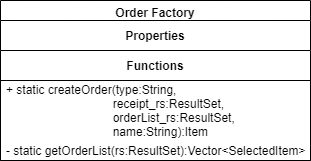
**Actors** – The FE controller, the DB controller, as well as the mySQL server being connected too.

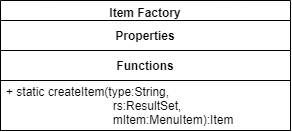
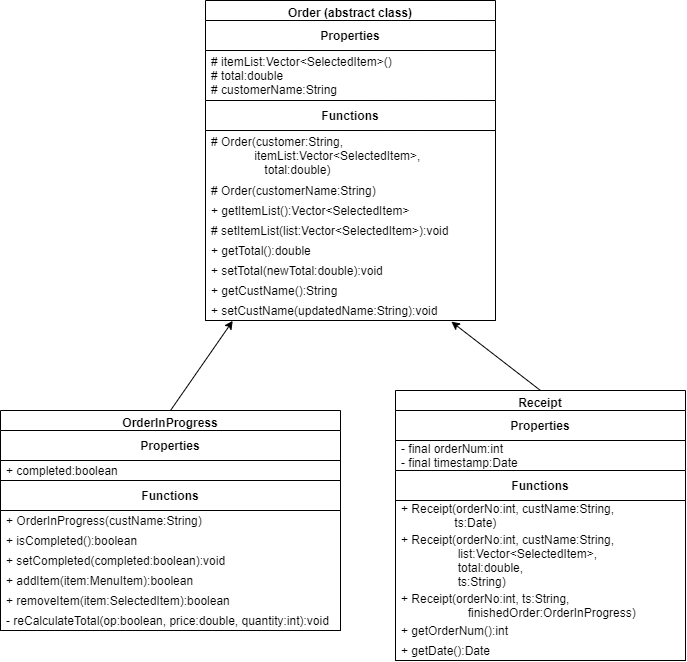
**Flow of Events -** FE controller requests info about a receipt -> DB Controller creates/executes query -> DB responds -> Passes found items and quantities to FE.

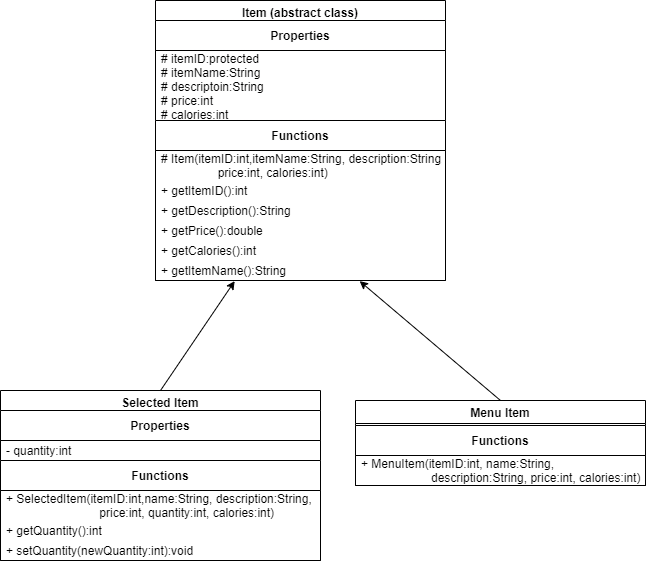
* 1. **Database Diagrams**
     1. **Database ER Diagram**
     2. **Menu Item Table Structure**
     3. **Receipt Table Structure**
     4. **Receipt Info Table Structure**

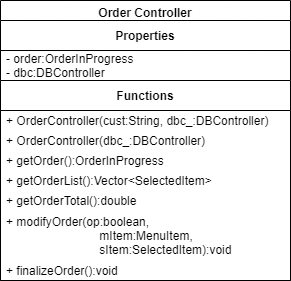
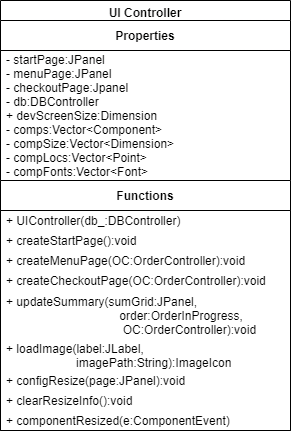
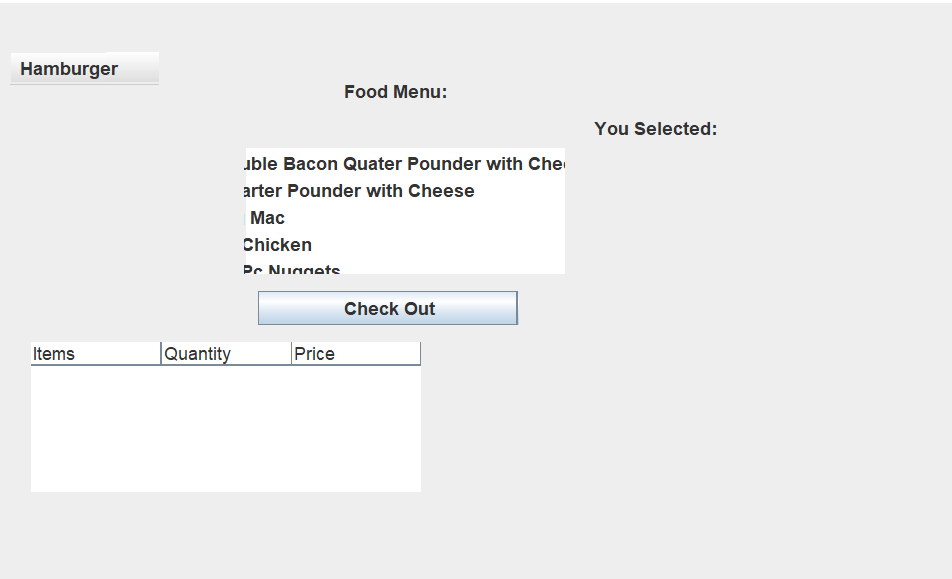
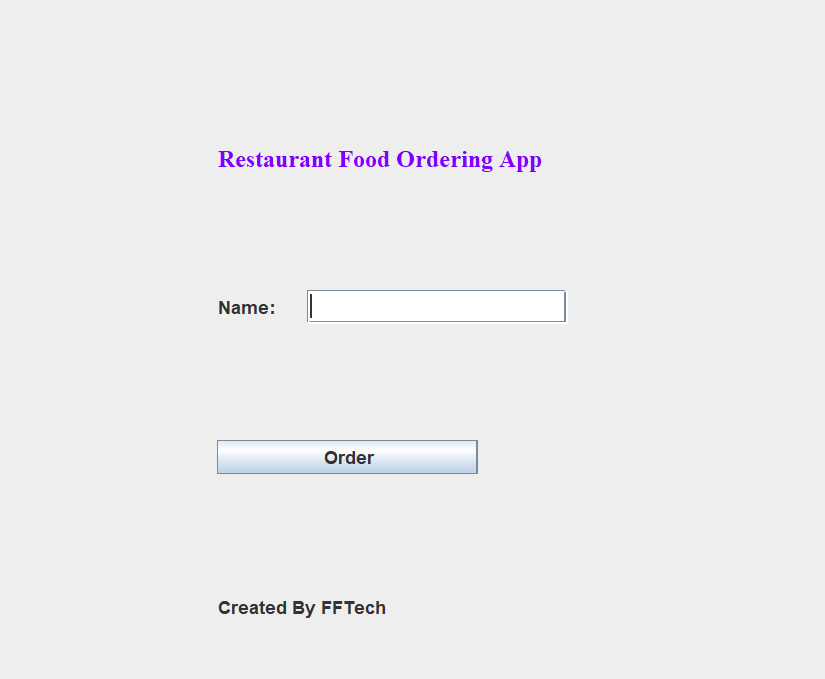
****

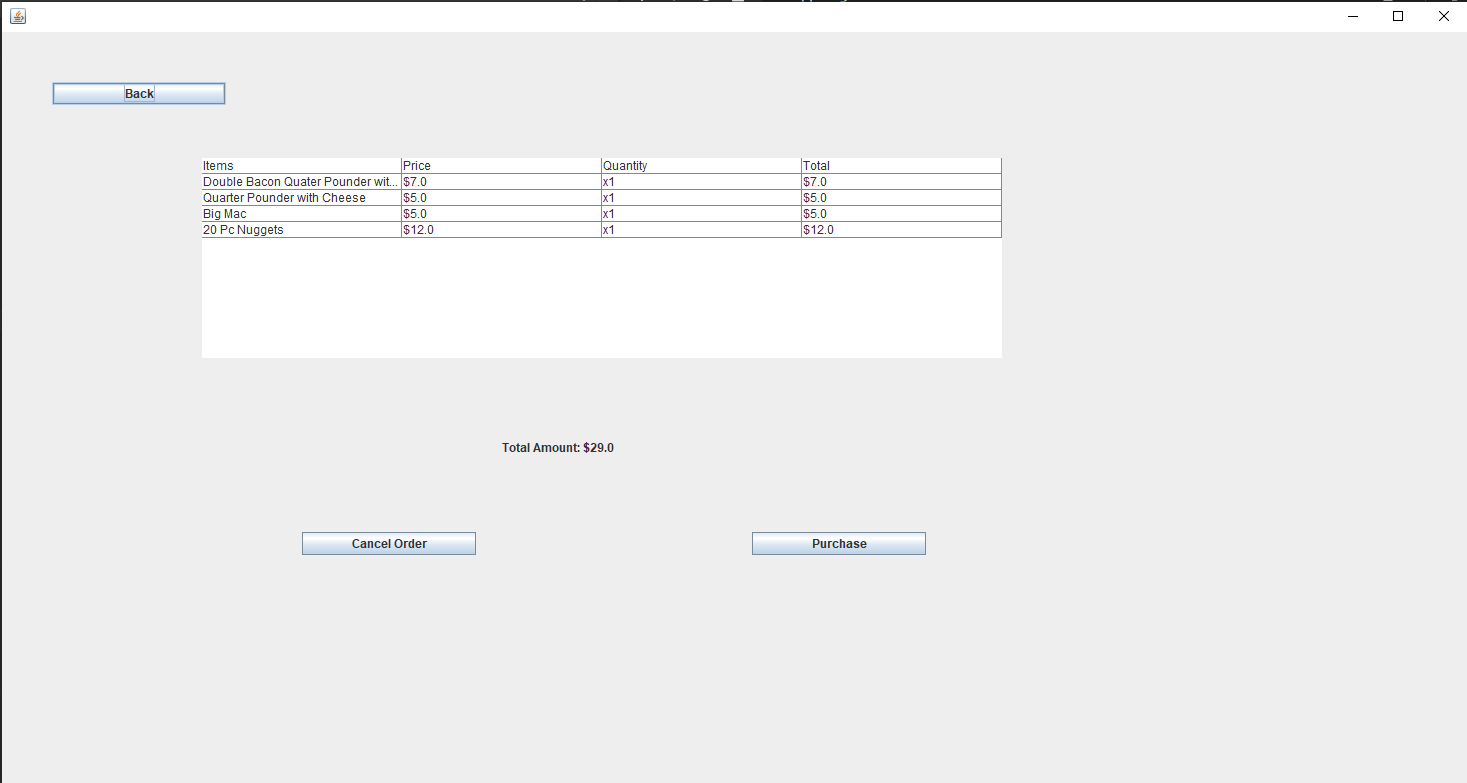
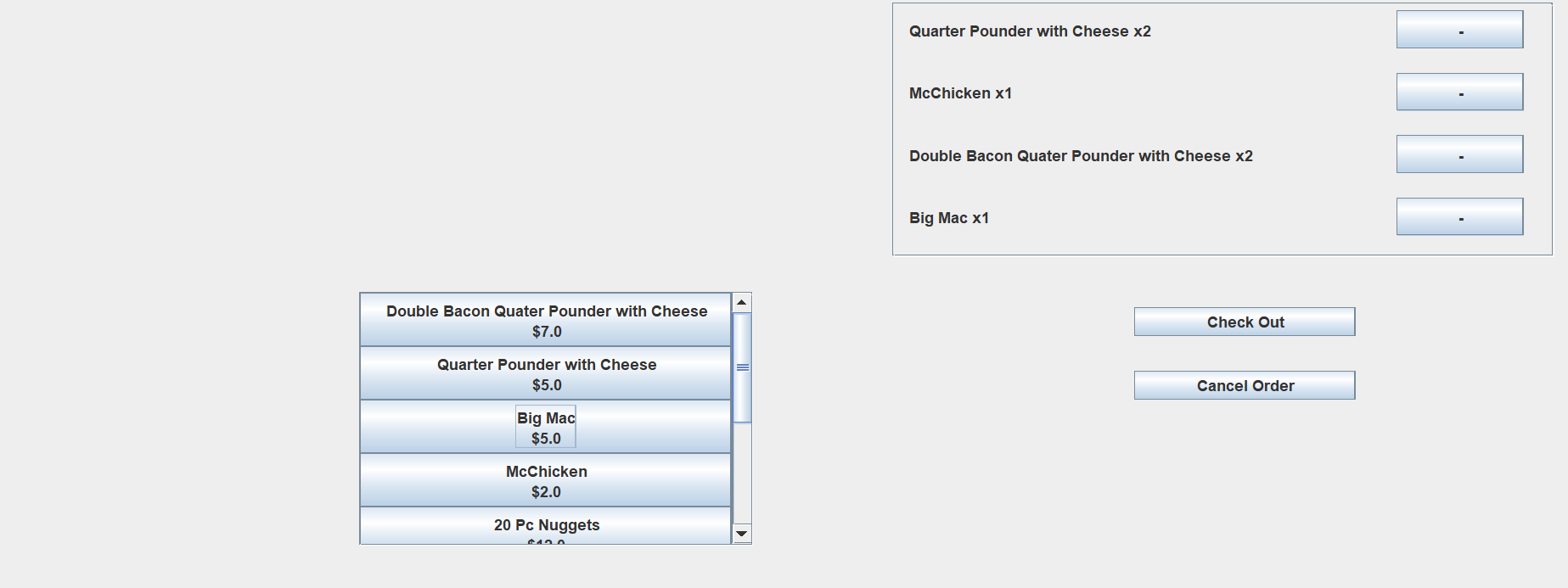
* 1. **Class Diagrams**
     1. **DBController.java**
     2. **Order Factory**

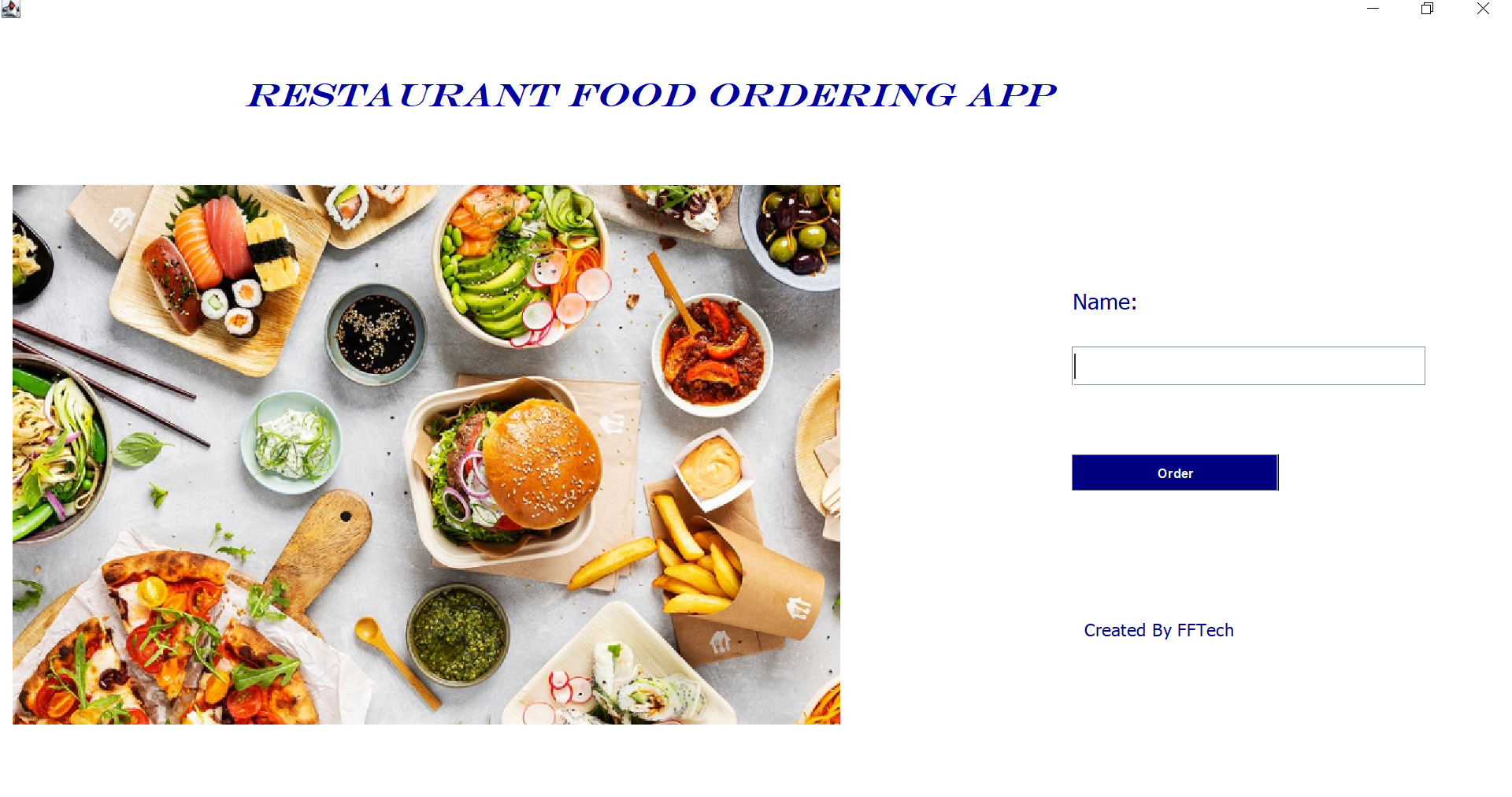
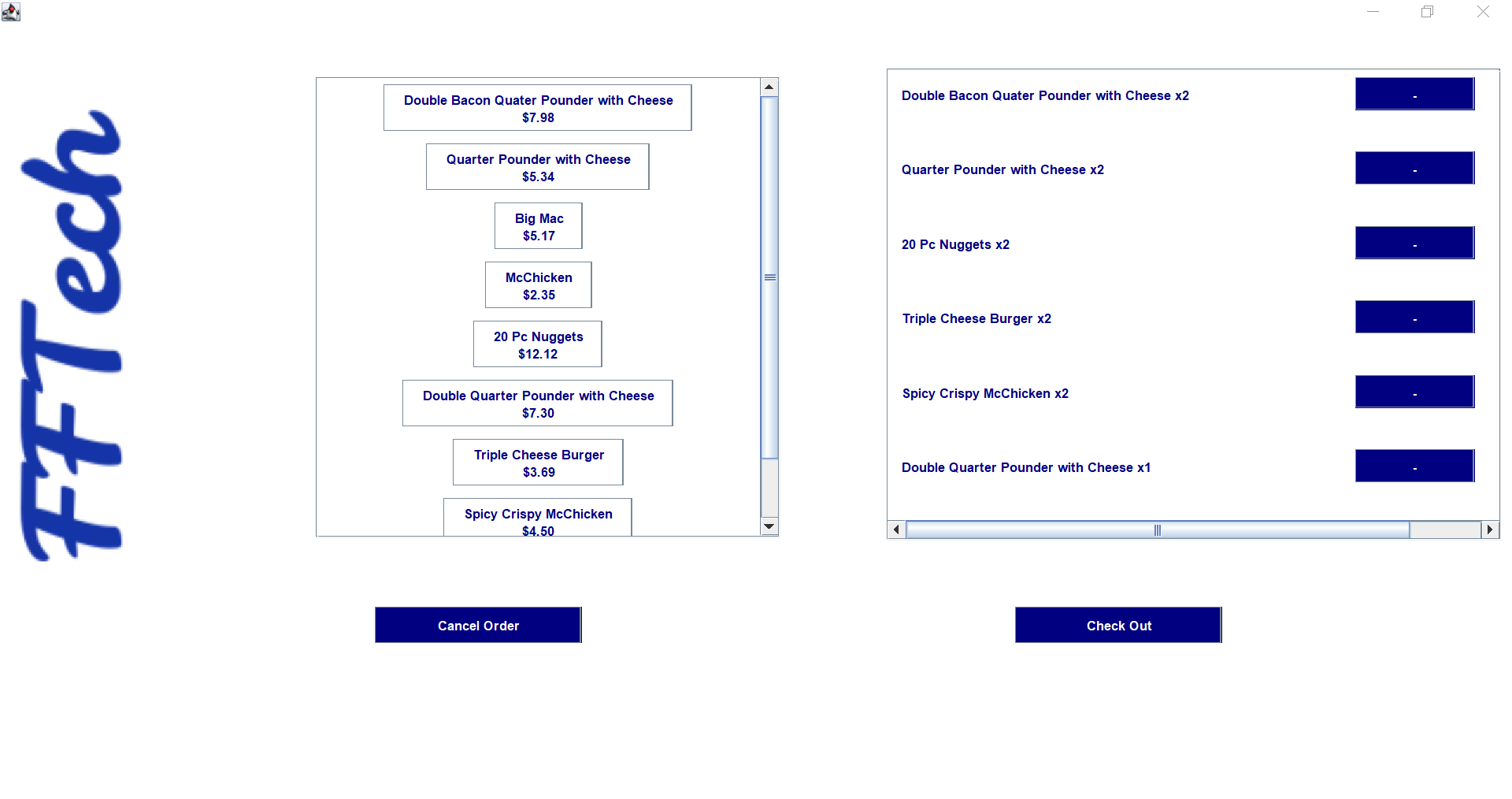
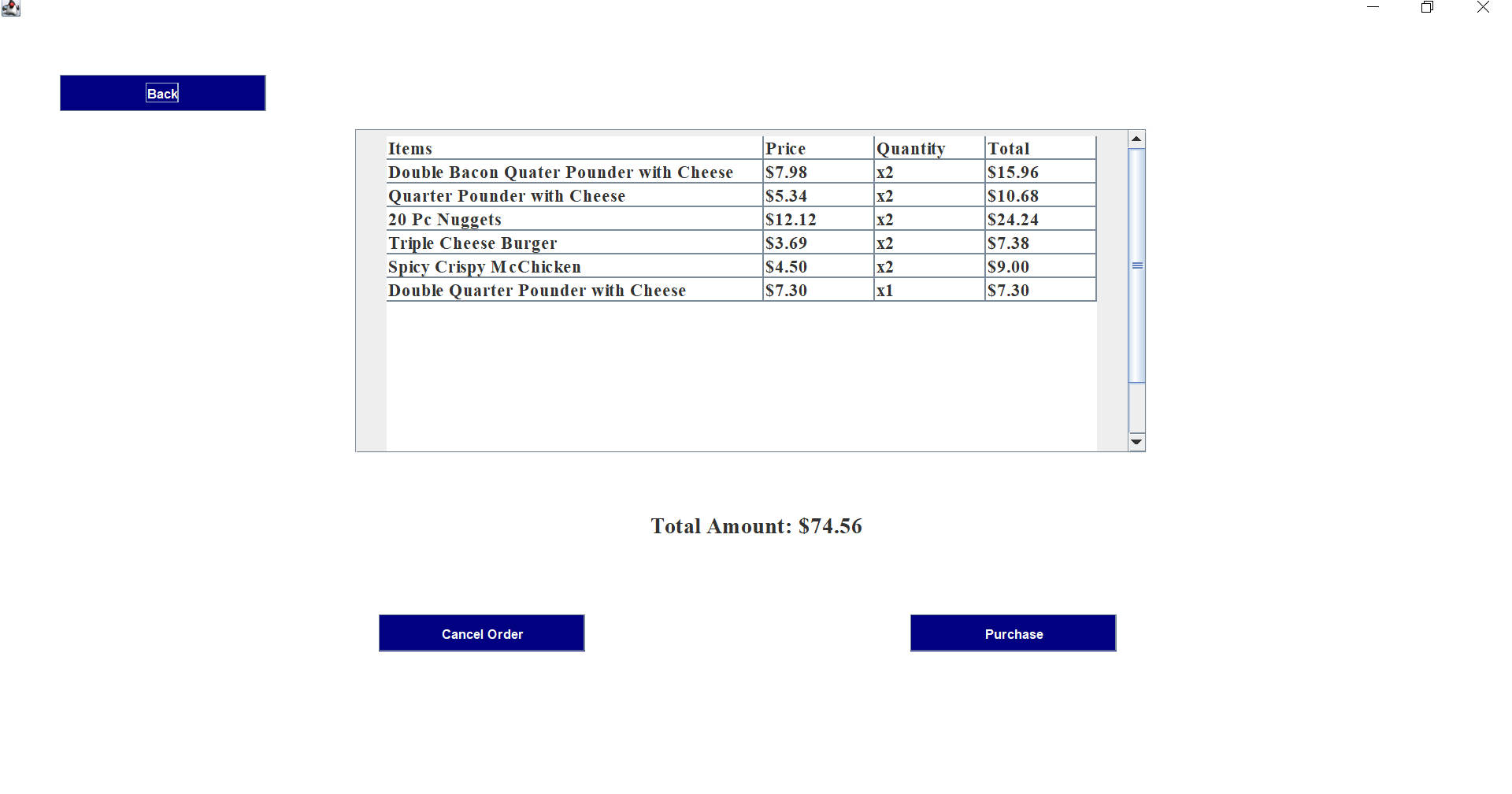
****

* + 1. **Item Factory**
    2. **Order.java**
    3. **Item.java**

****

* + 1. **OrderController.java**
    2. **UI Controller**
  1. **Change Log**
     1. **UI v1.0**
     2. **UI v2.0**

****

* + 1. **UI v3**
  1. **Software Evolution**
     1. **Database Evolution**
* **Database Controller version 1:**
  + **Functions to get:** 
    - **Menu Items for UI**
    - **Insert and save receipt information**
* **Database Controller version 2:** 
  + **Adding more functions to retrieve data:**
    - **Get receipt information**
    - **Get receipt item list information**
    - **As well as checks for valid DB connections.**
* **Future Additions:**
  + **Any other type of query or insertion can be added as a new function in the DB controller.**
    1. **Object Class**
* **Item/Order Classes version 1:**
  + **Abstract Classes set up with getter/setters.**
  + **Created two types of each class that extend the abstract base class, with additional attributes and getters/setters.**
* **Item/Order Classes version 2:**
  + **Create additional constructors for different ways to create each type of object. As well to facilitate factory patterns.**
    1. **UI Controller Evolution**
* **UI Controller version 1:**
  + **Loads menu from database**
  + **Creates checkout table and calculates total**
  + **does not fully interact with backend classes**
  + **displays UI in 1 page**
  + **Unable to insert receipts into the database**
* **UI Controller version 2:**
  + **Nicer UI**
  + **displays UI in 3 pages**
  + **Uses grid layout and JButtons instead of JList**
  + **Fully interacts with and uses all backend classes**
  + **able to insert receipts into the database**
* **UI Controller version 3:**
  + **Better looking UI, has colors and pictures**
  + **Window is now resizable and stay proportional and readable when resized**
  + **When the order is finished, it displays a successful order message and waits 2 secs, then goes to the start page to be used for the next order**

**Appendix A – Data Dictionary**

*<Data dictionary is used to track all the different variables, states and functional requirements that you described in your document. Make sure to include the complete list of all constants, state variables (and their possible states), inputs and outputs in a table. In the table, include the description of these items as well as all related operations and requirements.>*

**Appendix B - Group Log**

*<Please include here all the minutes from your group meetings, your group activities, and any other relevant information that will assist in determining the effort put forth to produce this document>*