Database Design Document

Team 64

JP Pham
Kevin Champagne
Connor Callan
Adidev Mohapatra
Ezra Lane

Executive Summary

The intent of project 2 is to create an automated POS system for Pom and Honey, a Mediterranean restaurant located in the MSC at A&M. This project will use Java for both the frontend and backend, and we will use AWS as the server for keeping track of all of our data. For our data, we will have three entities, Order, Product, and Item. Order will be used to keep track of what you think of on a customer's end when you are ordering something from a restaurant. It will keep track of what is actually in the order as well as the cost of the order and any discounts that are on the order as well. You have products, which are what goes into an order. The products consist of the food items that go into making that product, like rice, chicken, lettuce, etc. The product entity also keeps track of the portions of the items. Items are what the name sounds like, they are the individual items that make up an order. This is used in the database to keep track of how much of each item we have in inventory.

An order will consist of one or more products, and a product will consist of one or more items. The main interactions include adding items to products, adding products to orders, and using the information in an order to modify the inventory information of items. The most notable attributes are that the products will contain the actual pricing information, items will contain inventory quantities, and orders will be time-based (i.e. they will contain a date).

The assumption we made is that the customers can make only appropriate orders from the menu that correspond with the inventory. This means that they can't make orders that don't make sense but only orders the GUI and database can handle. The workers will also know exactly how to run the GUI and how to manage the inventory so that the database will be up to date. The databases and items that the GUI will run on will not fail and will always work as intended which is also one of the risk we took and assume could happen.

Purpose

Problem Statement: Pom and Honey is a restaurant at the TAMU MSC that serves Mediterranean food. Because of the volume of students Pom and Honey serves, the company needs an automated system for tracking sales and inventory. This system should be easily usable by employees and managers, with the highest priority being the system's speed. Without a system like this, Pom and Honey could easily run out of inventory, and managers would not have an easy way to know when to order more items.
Target Audience - Our target audience is the manager and workers. The managers want to see sales and oversee the trends and inventory of the restaurant for Pom and Honey. The workers will be using the GUI to interact with the customers, charging customers, adding in payment methods, declining payment methods, and such.
Proposed Solution - The proposed solution is to create a GUI system for the restaurant to manage customers' data such as orders, and for internal uses such as keeping inventory. The system will have two main client views, one for managers and one for servers. The server view will allow employees to take orders and store customer receipts. The manager view will allow managers to view sales and inventory at a high level.

High-Level Entity Design

Order

- **Purpose:** An order represents a list of items paid for at a specific time by a specific customer.
- **Justification:** The order is a necessary entity because it will allow the system to determine important metrics, such as average spending per order and the number of orders in a day. It will also allow the system to store, print, and retrieve receipts.
- **System:** The Order entity will take in the product entity as one to many meaning an order can have multiple products but the product entities can only connect to one order entity. The order will be what the customer will receive after paying for it. The customers can have multiple orders.

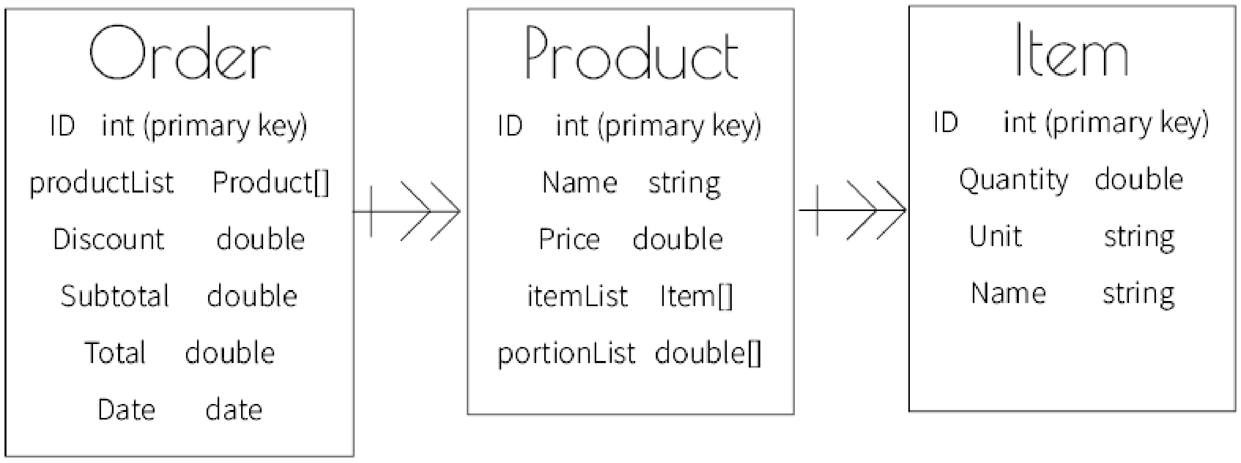
Product

- **Purpose:** A product represents a thing that can be added to an order. A product consists of items that modify the inventory.
- **Justification:** This entity represents the actual product that a customer buys. This entity could also help managers know what products are doing the best (ex: bowl vs gyro sales).
- **System:** The Product will take in the item entity as a one-to-many meaning the product entity can have multiple item entities but the item entity is only connected to one product. The product entity is essentially the combination of food that the customer ordered. Like if the customer ordered the bowl then the product is the bowl that contains many items within it.

Item

- **Purpose**: An item represents a change in the order that doesn't affect the cost of the order but does affect the inventory.
- **Justification:** The reason for Item is to separate from the other items that cost money but still keep track of the inventory for the modifiers.
- **System:** The items are the individual food that would add up to a product inside of something like a bowl or a gyro. The Item is connected to the product entity as one to many meaning the product can have multiple items while the item can only have a connection to one product.

Low-Level Entity Design



The UML diagram contains the three entities used for this project. Orders are the highest level entity, describing a list of products requested and paid for by a customer. A product represents a sellable entity, consisting of items. Finally, an item is anything that can go in a product and has a finite supply that must be tracked in the inventory.

Entity Attributes

“Order” holds all of the products that are being ordered as well as the prices and discounts of everything. Order keeps track of the products in that order (productList), any discounts on that order (Discount), the total without tax (Subtotal), the total with tax (Total), and the date of the order for bookkeeping (Date). “Product” represents the individual ingredients that go into making the food items. This entity keeps track of what product we’re making (Name), the price of the product and its toppings/extras (Price), a list of items that go into making the product as well as the toppings (itemList), and a parallel list of portions that go along with the items (portionList). “Item” is used for keeping track of all of the items in the inventory. You have the quantity of the item that we have in storage (Quantity), The unit for that quantity, for example milliliters or grams (Unit), and the name of that item (Name). All of the attributes have an “ID” attribute, this is used for bookkeeping purposes in case we need to go back and look at a specific order, product, or item.

Entity Relationships

First, an order has a list of one or many products. When a customer creates an order, it would not make sense to have zero products. Likewise, it would not make sense to limit the customer to one product (for example, one customer could buy two gyros, or a bowl and a drink). Next, a product has a list of one or many items. Note that for each item in a product, there is also a portion amount that is stored in a parallel array. We chose this relationship because items can have different portions in different products (ex: a bowl could have more rice than a gyro). It does not make sense for a product to be made of zero items, but it does make sense that a product could have more than one item in it.

Interactions

Interaction 1: Storing items in a product: When creating a product to add to an order, the system will need to know what items are in the product for inventory purposes. This will be done by storing two lists. The first list, "itemList," will store IDs to items included in the product. The second list, "portionList," will store the respective portions used in the items. Note that the units associated with each item are stored in the item entity itself. We made this decision because different items will be measured differently, but the same type of item should be measured in consistent units.

Interaction 2: Storing products in an order: This interaction is relatively simple: each product will act as a line item on the customer receipt. Once a product is created, it will be added to the order by storing its ID in the "productList" attribute.

Interaction 3: Orders updating the inventory of items: When an order is finalized, the item entities will need to be updated to reflect the new change in inventory. To do this, iterate through each product in the order. For each product, iterate through the item list and portion list. Look up each item in the database, and subtract the portion amount from the "Quantity" attribute. Upon completion of this algorithm, the database will have updated information on the inventory.

Assumptions and Risks

The assumptions we made are that the customers cannot have free refills, there are no discounts, customers can only buy from the available menu, and cannot buy the entire inventory. The risk is that the server for the database could fail and therefore the program that runs the database could crash (AWS could fail us). Another assumption is that the workers already know how to use the system and will be as efficient as possible and deal with orders from the customers. The workers will also only use the appropriate amount of food added to the product like one scoop of butter chicken or two scoops of butter chicken. The environment the GUI will run will be on an iPad or a tablet that will be connected to the university database that runs on AWS.