**Project Plan**

**Summary**

The project being built is called *Dan’s Bagel Shop*. It is a web application for a customer who would like to have an online presence for their bagel shop. This app would be usable by both customers and employees of the store.

Customers would use this app to order bagels, check on the status of their orders, and add money to an account that they would use to make purchases.

Employees would use this app to monitor and update orders, update inventory on-hand counts, and manage the overall menu items. There would be three subtypes of employee accounts:

* Chefs - primarily concerned with updating order preparation status and inventory counts
* Cashiers - primarily concerned with handling transactions with customers and updating order pickup status
* Owners/Admins - can perform all operations that chefs and cashiers can, as well as update the menu and hire/fire employees

**Team Organization**

Our team has three members: Jonas Knudsen, Tyler Beck, and Kaden Hellewell.

The job duties of each member may change as the project progresses, but here is what we are planning on working on:

Jonas Knudsen - Primarily focusing on programming in the back-end (i.e., database setup)

Tyler Beck - Primarily focusing on interfacing the front-end with the back-end

Kaden Hellewell - Primarily focusing on programming in the front-end (i.e., webpage / GUI design)

**Software Development Process**

The software development process for this assignment will be as follows:

* Requirements Gathering
  + Identify customers and writing down requirements
* High-Level Design
  + Designing major subsystems and interfaces
* Low-Level Design
  + Refine high-level design until pieces can be implemented
* Development
  + Write the code
* Testing
  + Unit tests - testing individual components (classes, functions, etc.)
  + System tests - testing the overall application as a whole
  + Regression tests - running unit tests after integration to make sure nothing new broke the old code
* Deployment
  + Install application for the user
  + In our case, we can just run the Django server locally and demonstrate functionality via Zoom screen sharing

In a real-world context (i.e., if this weren’t just a school assignment for one class for one semester), we would also have the following steps:

* Maintenance
  + Changes, additions, bug fixes, and enhancements
* Wrap-Up
  + Gather information about the project, what went well, what could have been done better

**Policies, procedures, and tools for communication**

We are using the Git version control system and the GitHub repository hosting

service.

In Git, whenever a change needs to be made, we will do the following:

* Create a new branch at the tip of `master`
* Make all changes necessary on this branch, pushing the commits to our shared GitHub repo
* Once all changes are validated as working, merge back into `master`

If we run into a merge conflict, the individuals who pushed up the code that conflicts

will work with each other to get that conflict solved (unless there's an obvious fix).

Our group’s primary form of communication is through a Discord chat room. (Discord is an online chatting service that provides text, audio, and video communication channels.) As COVID-19 is limiting the amount of in-person gathering that may take place, we found virtual meetings to suit our group much better than in-person meetings.

For our Scrum Board, we are using our group’s GitHub repository’s Projects feature to track issues. We have three columns: To Do, In Progress, and Done. These columns represent the state that each task is in.

**Risk Analysis**

1. **All Account Details**
   1. Each account must have an Email associated with it
   2. A first and last name must be provided for the account
   3. Passwords will be required
   4. All accounts will have a unique numeric ID
   5. Users will be able to change account info
      1. Passwords, name, and email

Risks:

There may be some invalid emails or other account details provided, but this should be mitigated by browser checks to ensure emails are valid. This is not a severe risk. If the account information is invalid, this will be caught at account creation. The user will have to create another account.

1. **Customer Account Details**
   1. $100 automatically added upon account creation
   2. Customers will be able to add funds to their account balance

Risks:

There is a mild to moderate risk of customer funds not properly being added or tracked. The severity of these risks would be very severe. Customers would not be happy to see their balance that their account had be completely wiped for no reason. In addition, if a balance was added for no reason to a customer’s account, the owner would not be happy (as they would have lost money). Some workarounds for this would to display a printable “receipt” page every time funds are added or subtracted from a user account. This would allow users to have definitive proof that a transaction actually occurred in case of a system failure.

* 1. The ability to place orders online
  2. View order history
     1. Reorder from order history
  3. View order status
     1. Orders may be canceled until bagel status is “Ready”
     2. Orders may **not** be changed once placed.

Risks:

There is a mild to moderate risk of orders falling out of the system or orders being duplicated. The severity of this risks would be very severe—customers would not be happy to see an order that they placed in advance not be ready for them to pick up when they expect. A workaround for these risks is to allow the admin / owner accounts to be able to create orders manually in the system. This would solve issues where customer orders were not properly created in the system.

1. **Chef Account Details**
   1. Will be able to see incoming orders
      1. Orders will be organized by pick up time
      2. The chef will be able to update the order status
      3. Orders will show up no later than 10 minutes before it needs to be ready

Risks:

There is a very low likelihood of risks happening with chefs being able to see incoming orders and change their statuses. If there was an issue here, the severity would be moderate. A workaround would be writing orders down by hand instead of having the system track the orders automatically (e.g., have a physical list of Post-It notes that the chefs can move around to sort by pickup time)

* 1. Chefs will be able to update the inventory
     1. remove items that have been used for an order
     2. remove items that have spoiled

Risks:

There is a very low likelihood of risks happening with chefs being able to update the inventory. If there was an issue here, the severity would be very mild. A workaround would be to have an admin manually count stock of various items in inventory and manually update the counts.

1. **Cashier Account Details**
   1. Be able to lookup customer’s numeric ID
   2. See list of orders and which customers they are associated with

Risks:

There is a very low likelihood of risks happening with cashiers being able to look up a customer’s numeric ID. The severity of an issue happening here would be minimal. If there was an issue with cashiers not being able to see the customer’s orders, the customer could show what they are seeing on their end to sort any discrepancies out.

1. **Admin (Owner) Account Details**
   1. Add and remove employee accounts
   2. Update the menu
   3. Order inventory

Risks:

There is a very low likelihood of risks happening with admins and owners being able to have overall control over the menu and employee accounts. If there was an issue, however, there would probably not be any workarounds until the issue was fixed server-side.

1. **Menu display**
   1. Able to be sorted by most popular items
      1. each item on the menu must keep track of how many times it was order in the last month
   2. Able to be sorted by price
   3. Default menu presentation will be sorted alphabetically within categories
      1. Menu categories are bagels, spreads, sandwiches, and beverages
   4. No premade combos – all items are sold individually

Risks:

There is a very low likelihood of risks happening with being able to sort the menu. The severity of any of these issues would be very light. If the menu had any issues with displaying anything, the customers could simply call or enter the store and order things the “old fashioned” way.

1. **Orders**
   1. Pick up times for an order may be specified
      1. Orders can only be placed one week in advance
      2. Orders without a specified pickup time will have a pickup time of when the order was placed plus 10 minutes
   2. Orders not picked up within 30 minutes of being ready are discarded

Risks:

There is a very low likelihood of risks happening with being to specify order pick up times. The severity of these issues would be mild – if there was any issue with specifying order pickup times over the web app, users could call into the store instead. There wouldn’t be any major consequences from these pickup times being incorrect.

1. **Inventory**
   1. Items ordered for are the same as items sold on the menu
   2. Each item can be ordered individually

Risks:

There is a very low likelihood of risks happening with items being ordered individually. Customers ordering items individually is very clear. The severity of any issues would be very mild, but there wouldn’t really be any workarounds to this besides providing more clear information about how the menu works.

* 1. Provide an analytics screen
     1. See what items need to be ordered
     2. See the sale figures of each item

Risks:

There is a moderate risk of the analytics being incorrect. The severity of these issues would also be moderate – the admin looking at these figures might make incorrect purchasing decisions and advertising decisions. A workaround for having incorrect analytics would be to have the admin manually comb through all orders and make decisions manually in

1. **Existing Customers**
   1. Integrate existing customers into the new system
      1. Waiting for details, currently unknown how this existing database looks

Risks:

There is a very low risk of migrating existing customers to the new system. However, if we missed a customer in this process, the consequences would be very severe – a customer with an existing balance may lose everything in the switch. One workaround would be to contact all customers before and after the switch to alert them (and follow up with them) about the change. If they had any specific issues with the migration then the admin could add their account information manually.

**Configuration management plan**

Refer to the README.md for the configuration management plan.