1. What features were implemented?

View - Within our project we were able to get a basic website up and running. On the home page, the user can click the Add List button. They are redirected to the Add list form. After the user enters a name, the system creates a new list with the specified name. The user can click on individual lists to get a closer view of a specific list. Within this view, the user can add a single item at a time to the list or delete the list entirely. They can also select a specific item and the system will highlight the item.

We also implemented and verified several features on the server side of the system. This includes creating and changing the password, checking if the user exists, logging in, and changing the user display name. We also implemented a Pantry class so the server could handle Pantry requests.

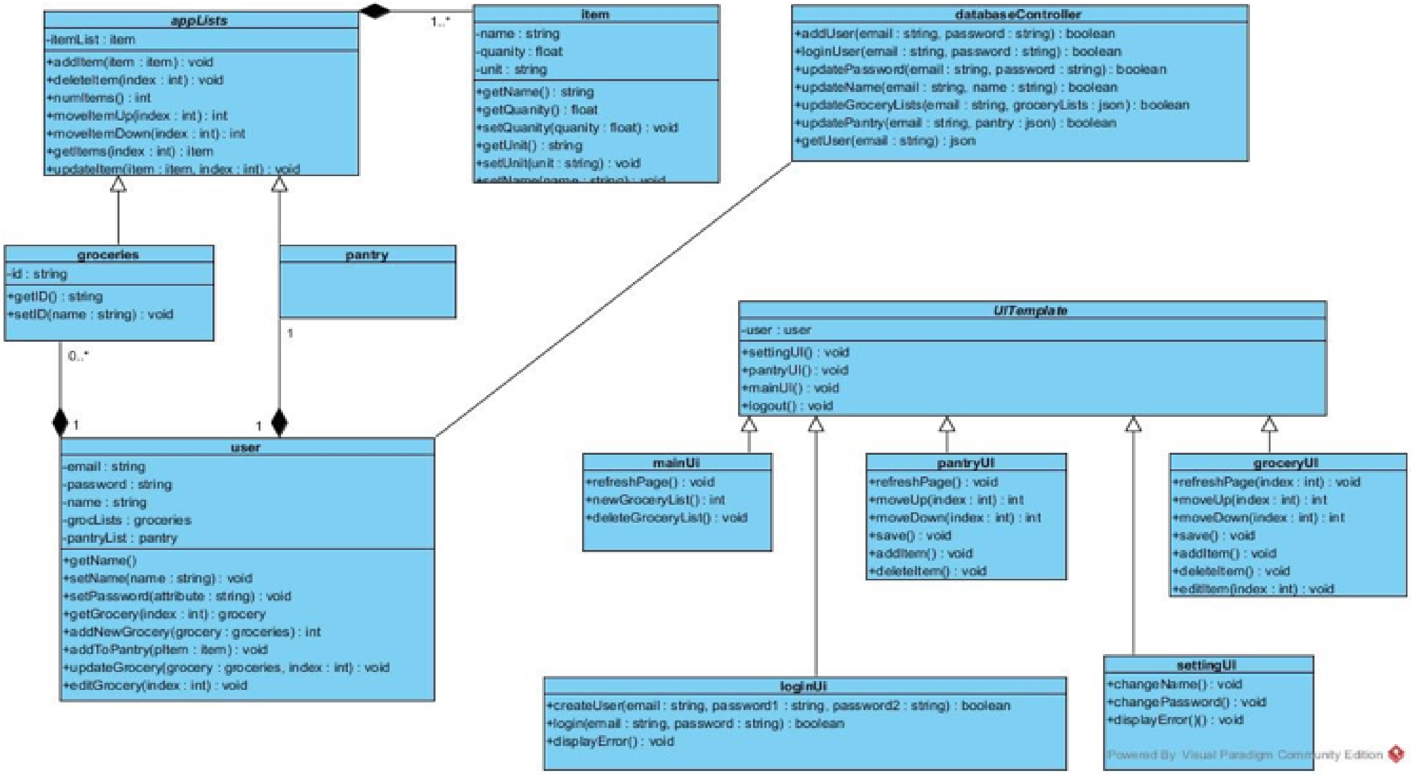
2. Which features were not implemented from part 2?

There were some features not implemented on the frontend that were specified in part 2. Many of these features were available in our User, Grocery, and Pantry classes. We did not implement the Authentication and Log in features specified in Part 2 on the front end of our website. This includes, but is not limited to: creating a new account, logging into the account, creating changing the password, changing the display name, etc. We were able to test and verify, with the use of unit testing, that our backend class were capable of implementing these abilities but were unable to implement them on the front end of our project.

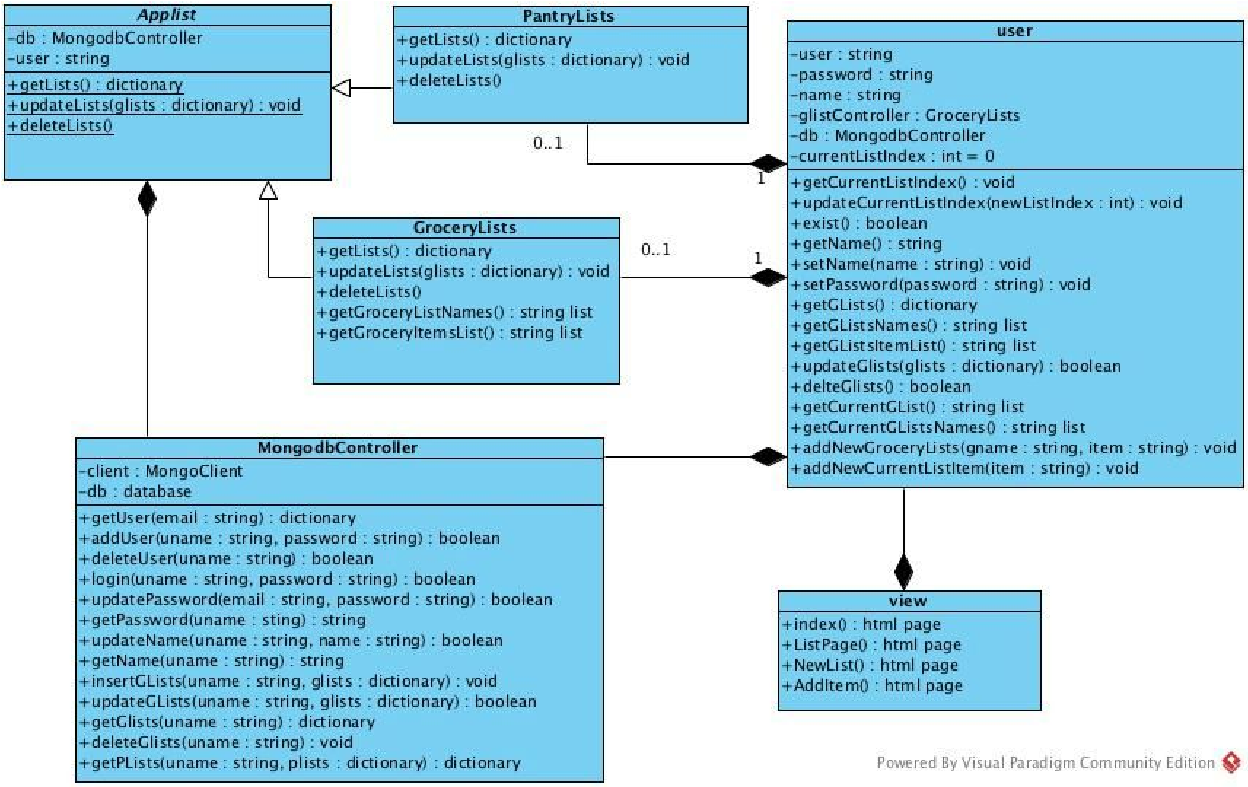
On the front end, we did not implement the Settings page which encompasses all the User Settings topic of our user requirements, including selecting a display name, changing password, etc. On the specific grocery list page, we did not implement the ability to edit or delete items, sort items. We also did not implement the Pantry page.

3. Show your Part 2 class diagram and your final class diagram. What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

Project Part 2 Class Diagram


appLists
-itemList : item
+addItem(item : item) : void
+deleteItem(index : int) : void
+numItems() : int
+moveItemUp(index : int) : int
+moveItemDown(index : int) : int
+getItems(index : int) : item
+updateItem(item : item, index : int) : void
1..*
item
-name : string
-quanity : float
-unit : string
+getName() : string
+getQuanity() : float
+setQuanity(quanity : float) : void
+getUnit() : string
+setUnit(unit : string) : void
+setName(name : string) : void
databaseController
+addUser(email : string, password : string) : boolean
+loginUser(email : string, password : string) : boolean
+updatePassword(email : string, password : string) : boolean
+updateName(email : string, name : string) : boolean
+updateGroceryLists(email : string, groceryLists : json) : boolean
+updatePantry(email : string, pantry : json) : boolean
+getUser(email : string) : json
groceries
-id : string
+getID() : string
+setID(name : string) : void
pantry
1
0..*
1
1
user
-email : string
-password : string
-name : string
-grocLists : groceries
-pantryList : pantry
+getName()
+setName(name : string) : void
+setPassword(attribute : string) : void
+getGrocery(index : int) : grocery
+addNewGrocery(grocery : groceries) : int
+addToPantry(pItem : item) : void
+updateGrocery(grocery : groceries, index : int) : void
+editGrocery(index : int) : void
UITemplate
-user : user
+settingUI() : void
+pantryUI() : void
+mainUI() : void
+logout() : void
mainUi
+refreshPage() : void
+newGroceryList() : int
+deleteGroceryList() : void
pantryUI
+refreshPage() : void
+moveUp(index : int) : int
+moveDown(index : int) : int
+save() : void
+addItem() : void
+deleteItem() : void
groceryUI
+refreshPage(index : int) : void
+moveUp(index : int) : int
+moveDown(index : int) : int
+save() : void
+addItem() : void
+deleteItem() : void
+editItem(index : int) : void
loginUi
+createUser(email : string, password1 : string, password2 : string) : boolean
+login(email : string, password : string) : boolean
+displayError() : void
settingUI
+changeName() : void
+changePassword() : void
+displayError()() : void
Powered By Visual Paradigm Community Edition

Final Diagram


Applist
-db : MongodbController
-user : string
+getLists() : dictionary
+updateLists(glists : dictionary) : void
+deleteLists()
PantryLists
+getLists() : dictionary
+updateLists(glists : dictionary) : void
+deleteLists()
0..1
1
user
-user : string
-password : string
-name : string
-glistController : GroceryLists
-db : MongodbController
-currentListIndex : int = 0
+getCurrentListIndex() : void
+updateCurrentListIndex(newListIndex : int) : void
+exist() : boolean
+getName() : string
+setName(name : string) : void
+setPassword(password : string) : void
+getGLists() : dictionary
+getGListsNames() : string list
+getGListsItemList() : string list
+updateGlists(glists : dictionary) : boolean
+delteGlists() : boolean
+getCurrentGList() : string list
+getCurrentGListsNames() : string list
+addNewGroceryLists(gname : string, item : string) : void
+addNewCurrentListItem(item : string) : void
GroceryLists
+getLists() : dictionary
+updateLists(glists : dictionary) : void
+deleteLists()
+getGroceryListNames() : string list
+getGroceryItemsList() : string list
0..1
1
MongodbController
-client : MongoClient
-db : database
+getUser(email : string) : dictionary
+addUser(uname : string, password : string) : boolean
+deleteUser(uname : string) : boolean
+login(uname : string, password : string) : boolean
+updatePassword(email : string, password : string) : boolean
+getPassword(uname : sting) : string
+updateName(uname : string, name : string) : boolean
+getName(uname : string) : string
+insertGLists(uname : string, glists : dictionary) : void
+updateGLists(uname : string, glists : dictionary) : boolean
+getGlists(uname : string) : dictionary
+deleteGlists(uname : string) : void
+getPLists(uname : string, plists : dictionary) : dictionary
view
+index() : html page
+ListPage() : html page
+NewList() : html page
+AddItem() : html page
Powered By Visual Paradigm Community Edition

UI Changes

For UI changes, instead of using the UI template with inherited templates for each page, we instead created a view that handled multiple html pages. This changed because, at the time of design, we were unfamiliar with webpage design. As we learned how to use Flask, we found various limitations that did not allow us to implement our original design. Additional methods were created for the User class to allow Flask to render the web pages the way we wanted.

Controller Changes

The user class was our primary controller class. The UI only interacted with the user class to communicate with the database.

Model Changes

The model had minor changes. The main changes were the names of the method to match the overall design flow

4. Did you make use of any design patterns in the implementation of your final prototype? If so, how? If not, where could you make use of design patterns in your system?

In our final prototype, we made use of the MVC design structure. We used an observer pattern where the observer (Flask methods). Our user class used a strategy controller to send messages to our db Model. Though we did create views, we did not implement the Composite pattern as part of this design structure.

5. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

Through this process, we have learned that analysis and design is very helpful in giving a visual end goal to achieve. By defining what we need to do early, we can save time spent later that would be wasted using a try-implement-redo method. However, we have also learned that not everything will go according to plan during the course of a project. Timelines and goals will have to change due to unforeseen circumstances. Also, development work involves a lot of trial and error and learn by doing. Documentation for various languages and modules can be limited will not always be widely available online. Therefore there can be considerable time spent testing methods and trial and error as part of that learning process. Time must be allocated for this so that projects can meet their projected deadlines. We have learned about the various ways to design and implement a system that we can hopefully utilize in the future for other projects. We also discovered that test design development can be instrumental in debugging issues when connecting the front and backend.