Final Report
CS 157A, Fall 2019
Intro to Database Management System

StoreFront Web Application
Team 30

Aaron Warren, Phu Tran, Evan Ugarte

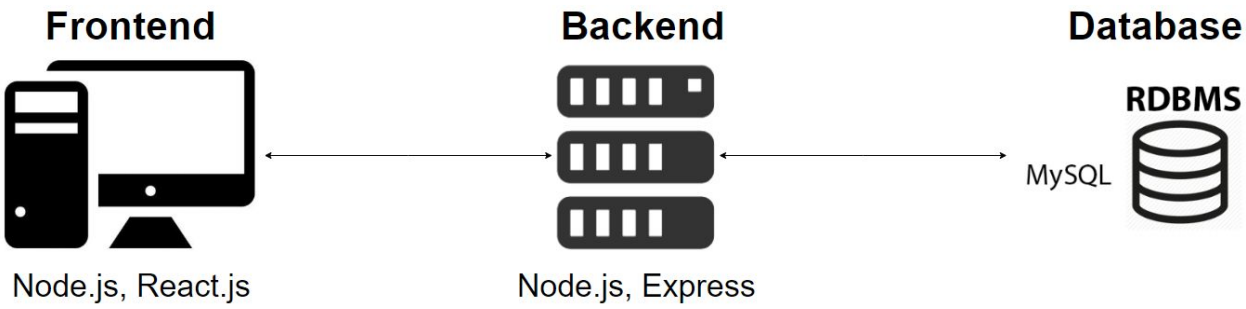
Table of Contents

Project Requirement

Project Description

This application would be takes the place of a physical cashier, where users would be able to order food from a web app and pick their order up when it is ready. The application is able to manage members, their credit card information, food inventory stock levels and item prices through a database. The stakeholders of this project are company owners who have a kitchen that takes orders, programmers who implement the application, project managers of the project, and store managers. The project is important as a business can use this application to take the place of a worker, and instead hire more cooks to handle demand. This project is important as it streamlines the process of ordering food at a business. Customers will be able to clearly select food choices based on photos and descriptions, and pay for their food by credit card.

System Environment

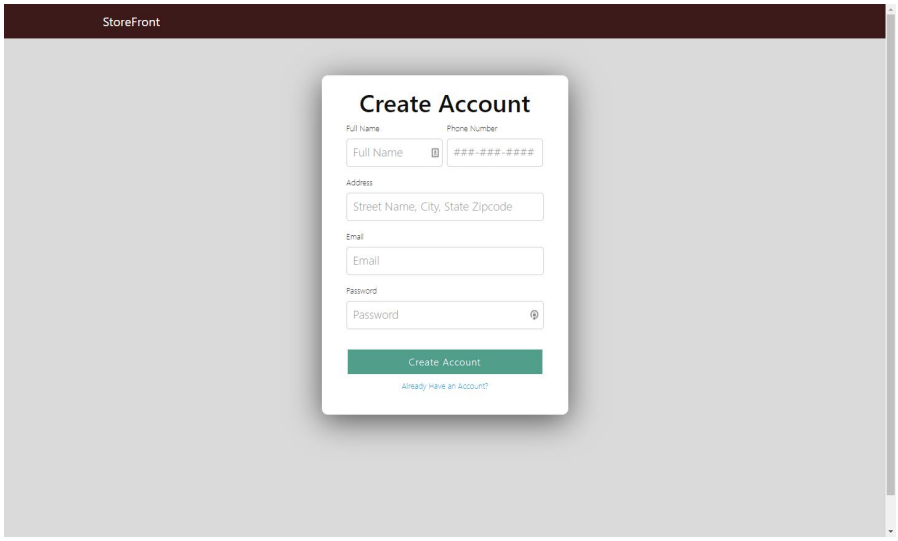


For our system environment, we tried to make it as linear as possible to make it easy to import into an AWS ec2 instance for fully online hosting. Through this we decided to host our database on an AWS server that was using MySQL. For our front and backend we decided to program in JavaScript due to high proficiency in the language from one of our team members. Through this we used NodeJS as the base framework to host our server and then ReactJS for rendering and frontend logic due to its high number of users which could help with various debugging if any issues cropped up throughout the lifetime of our project.

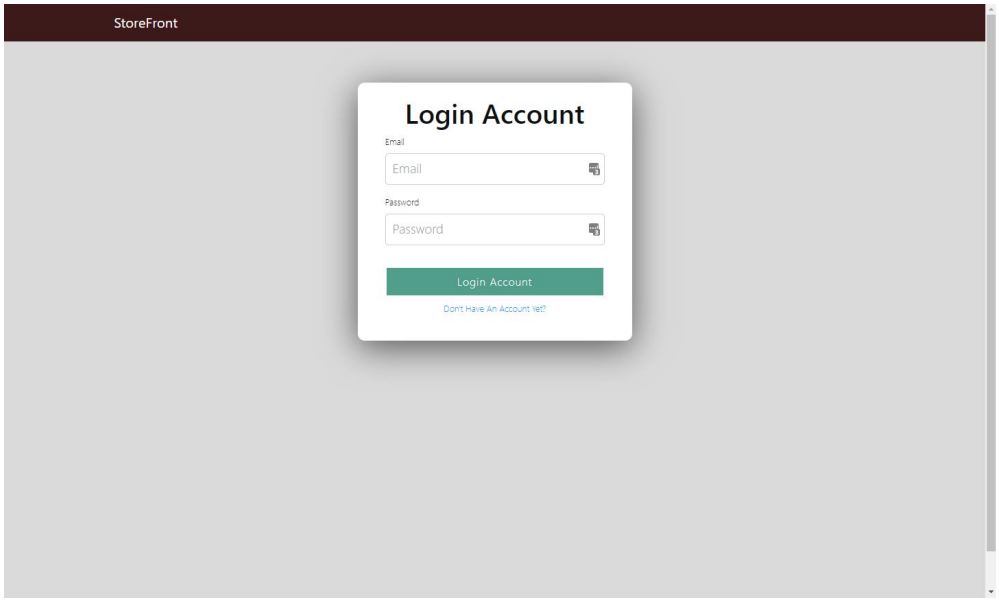
Functional Requirements

The functional requirements of our project are what anyone would expect from an online marketplace. These include:

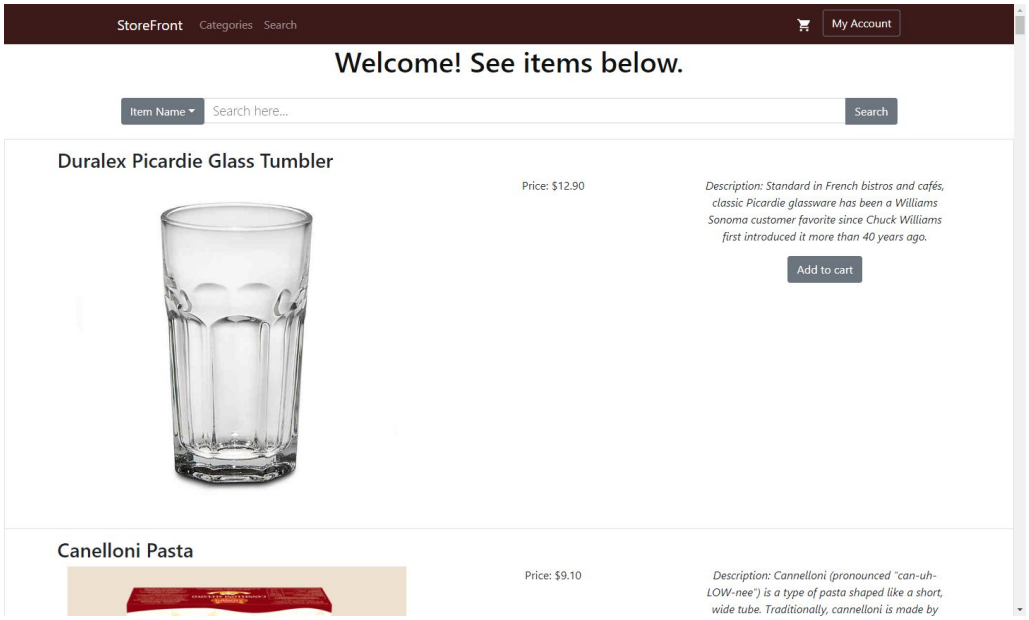
- Creating an account
    - Users will be able to create their account by entering their email, password, and additional information such as name. For those who already have an account and try to sign-up for a new one, the website will display a message alerting the user that their email is already in use for an existing account.
    - The system will check if a user's email and username already exists in the database. If email/username wasn't found, the system will write the new email and password to the database and display a message to the users letting them know that they have successfully created an account. If an email exists or is invalid, the system will display an error message to let users know that their account was not able to be created.



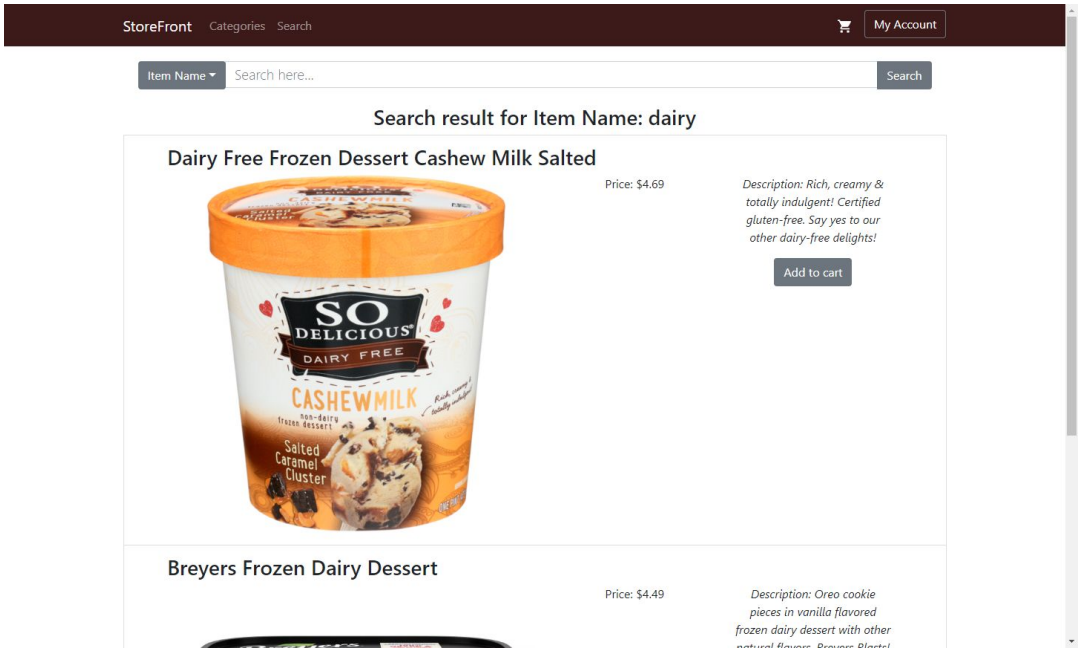
- Login
    - Users can login to the webpage by entering their registered email as the username and password.
    - The system will check to see if the given email and its associated password are the same with the information that are stored in the database. If it is, the system will allow the user to access their account. If it is not, the system will display an error message through the webpage to let the user know if the email or password is incorrect.


StoreFront
Login Account
Email
Email
Password
Password
Login Account
Don't Have An Account Yet?

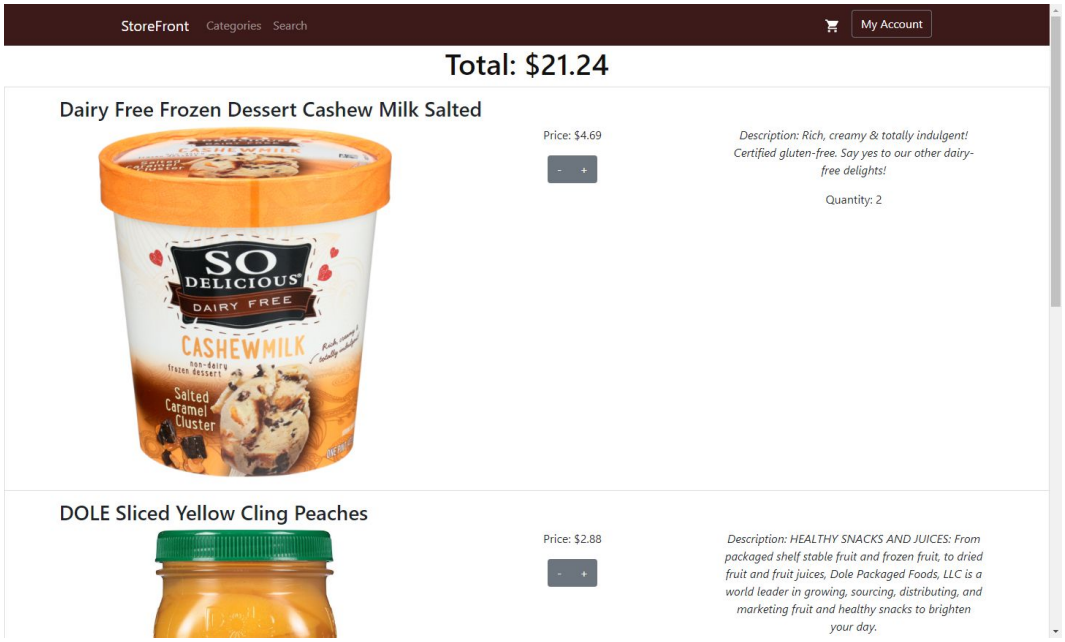
- Browse Items
    - Users will be able to view all of the items in the database through the StoreFront webpage. Each item will have an associated image, name and price.
    - To display the items to the user, the system will access the item entity set in the database and return all of the values in the table.



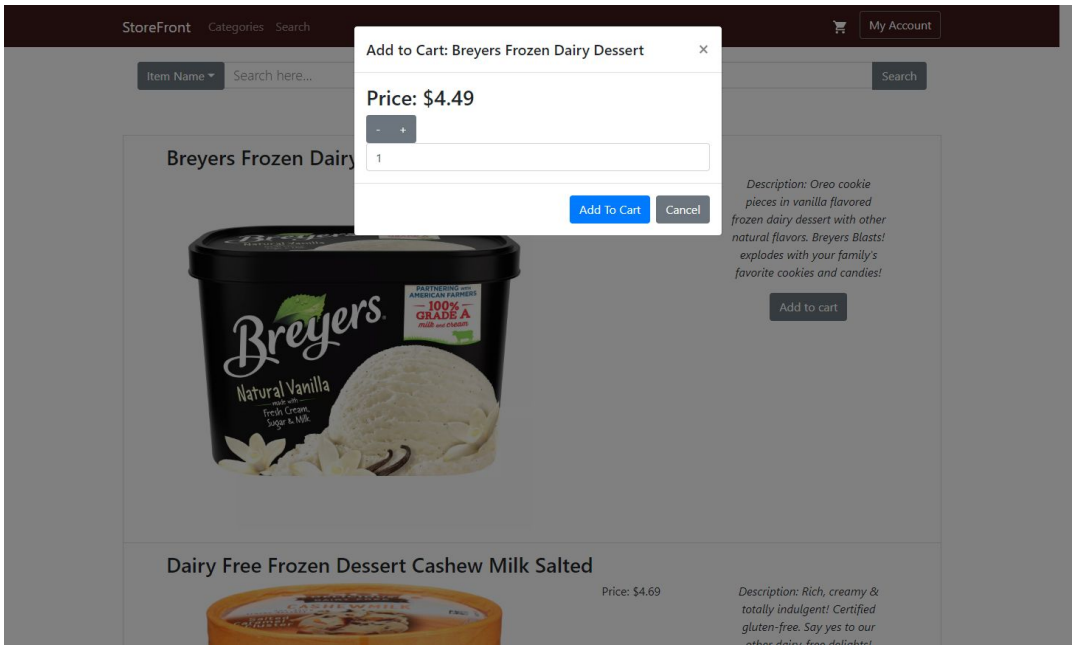
- Search Items
    - Users will be able to use system's search tab to search for specific items by item fields, such as category or name.
    - After a search is entered, the system will access the database to find all appropriate items. The results of the query will be displayed to the user. If the query returned a total of zero items, then a message will be displayed to let the user know.



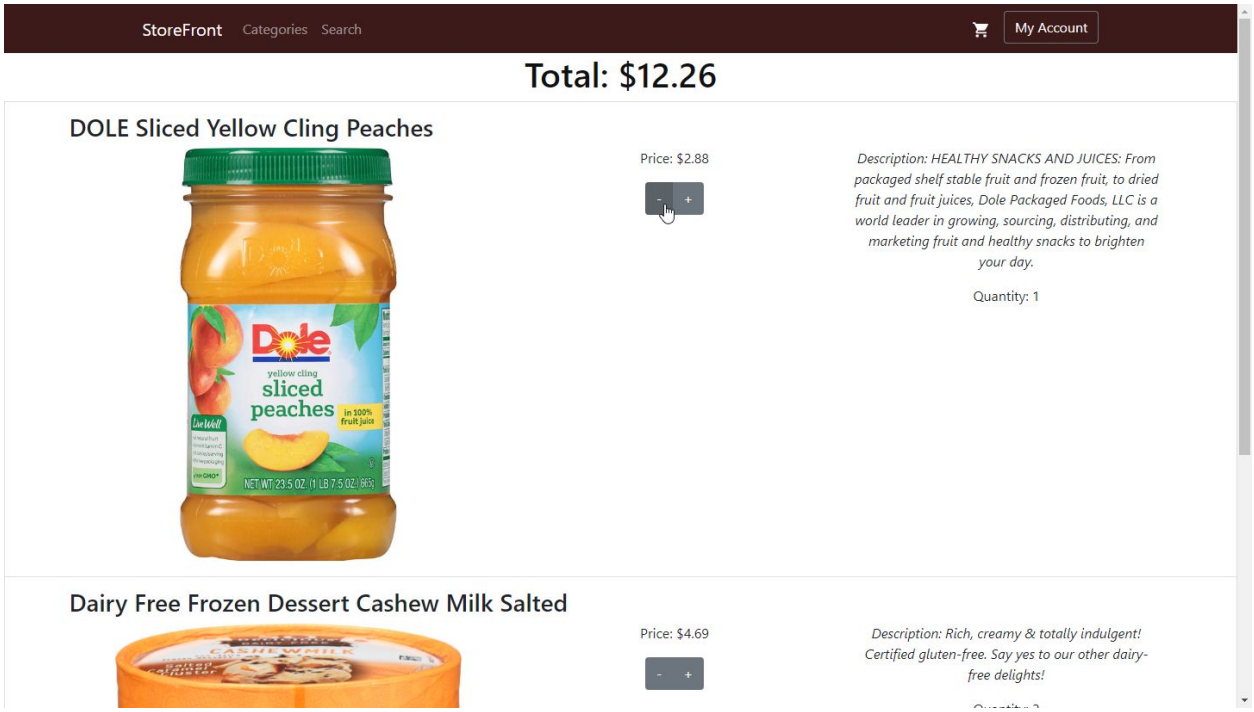
- View cart
    - The system will display a cart button to the user which upon being clicked will redirect to the cart page.
    - Users will be able to see all items added to the cart plus the total price.

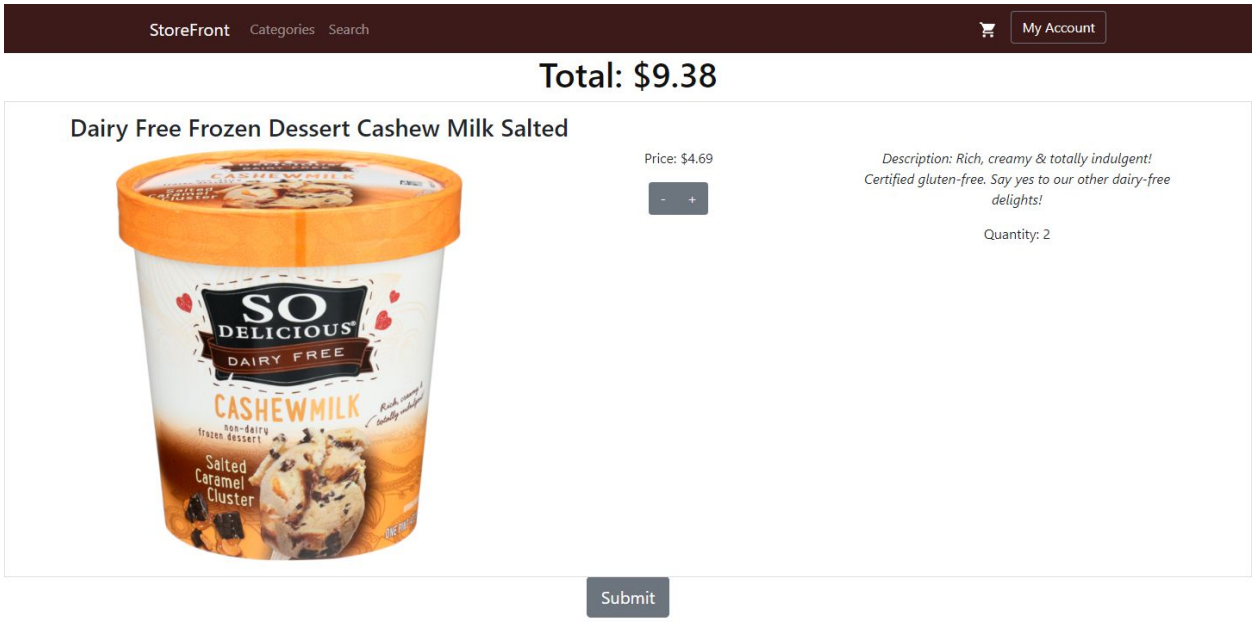


- Selecting and adding items to the cart
    - Users can select the items that they are interested in and add them to the cart to purchase later.
    - The system will keep track of current items in the cart for later use and only remove when the user deletes or purchases the item from the shopping cart.

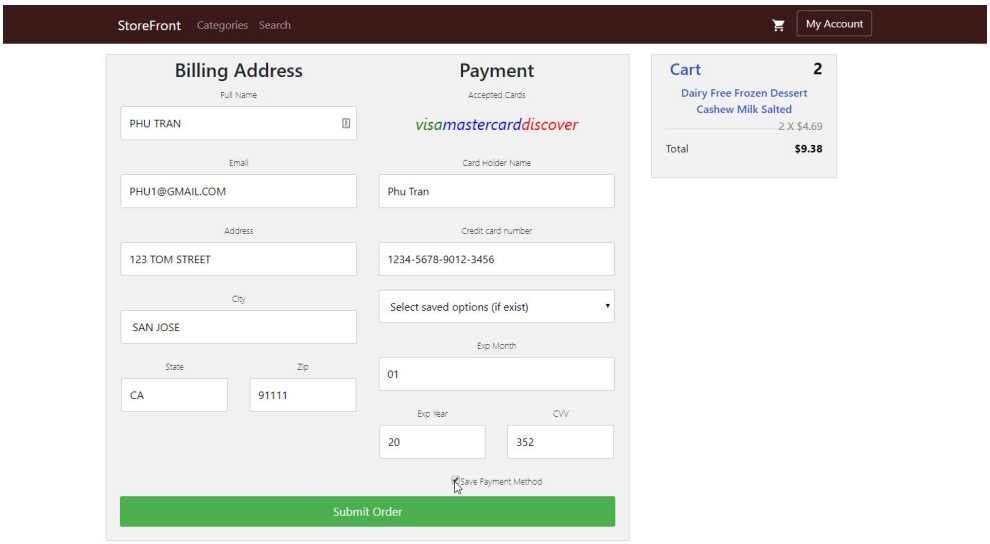


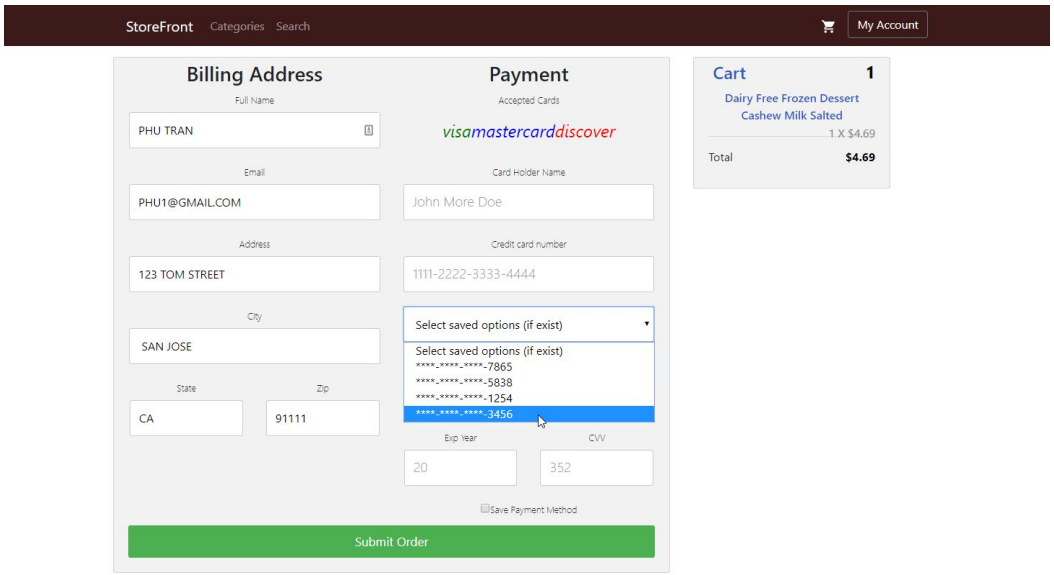
- Deleting items in the cart
    - Users can remove/delete any existing item inside their shopping cart that they no longer want to purchase.
    - The System will access a particular relation in the database and delete the entity of the item(s) that the user specified to remove.



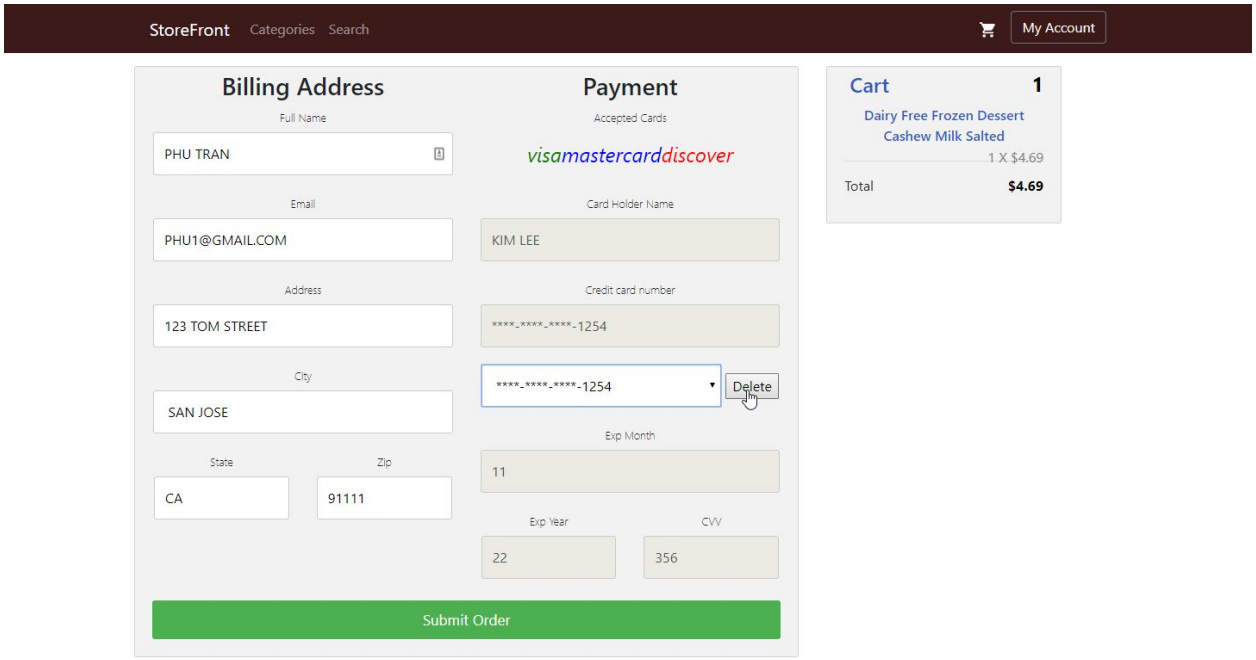


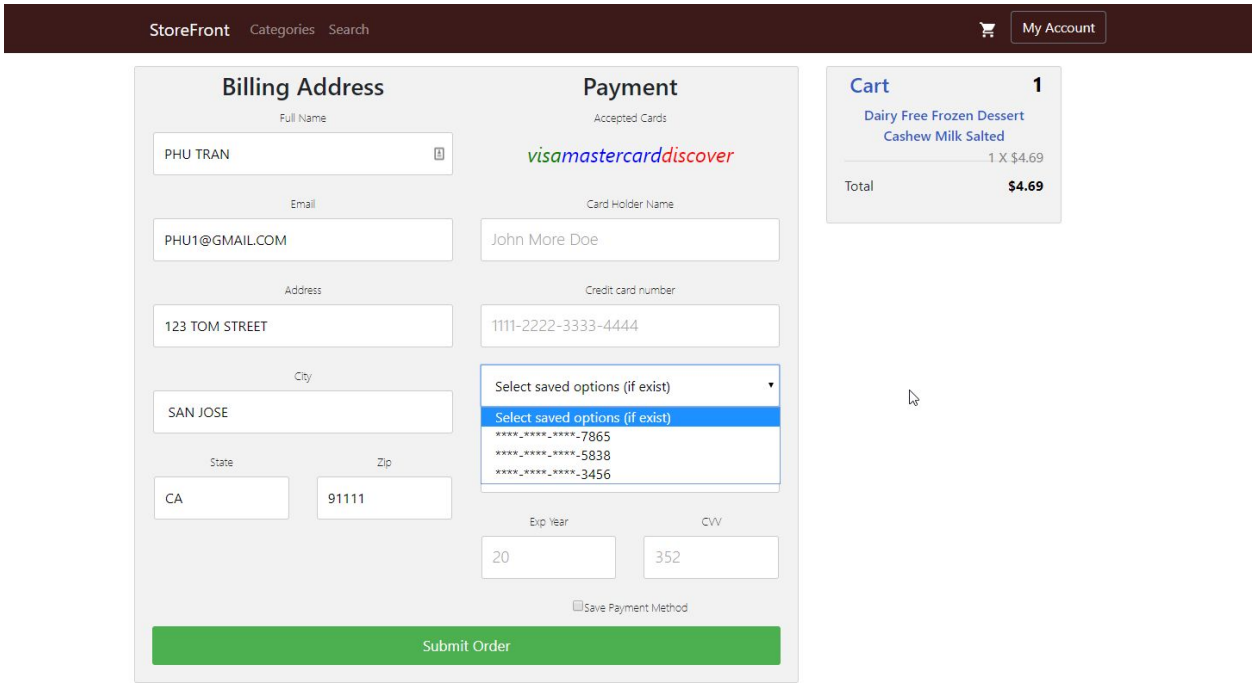
- Adding new payment methods
    - Users will be able to add a new credit card as a method payment by entering the card number, card type, holder name, CVV (card verification value), and card expiration date.
    - System will check with the database to see if the card number already exist or not. If so, the user will be notified that this payment method already exists. If it does not, store all this information in a secure way in the database



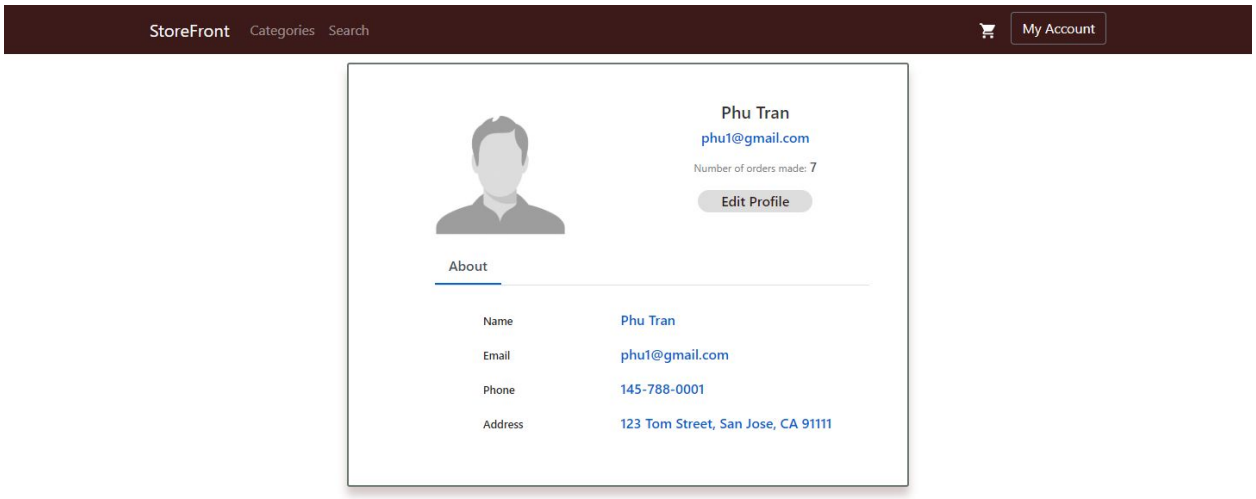


- Deleting payment methods
    - User will be able to delete exist payment method of their choice.
    - System will access the database and delete the entity that match the value that the user selected.

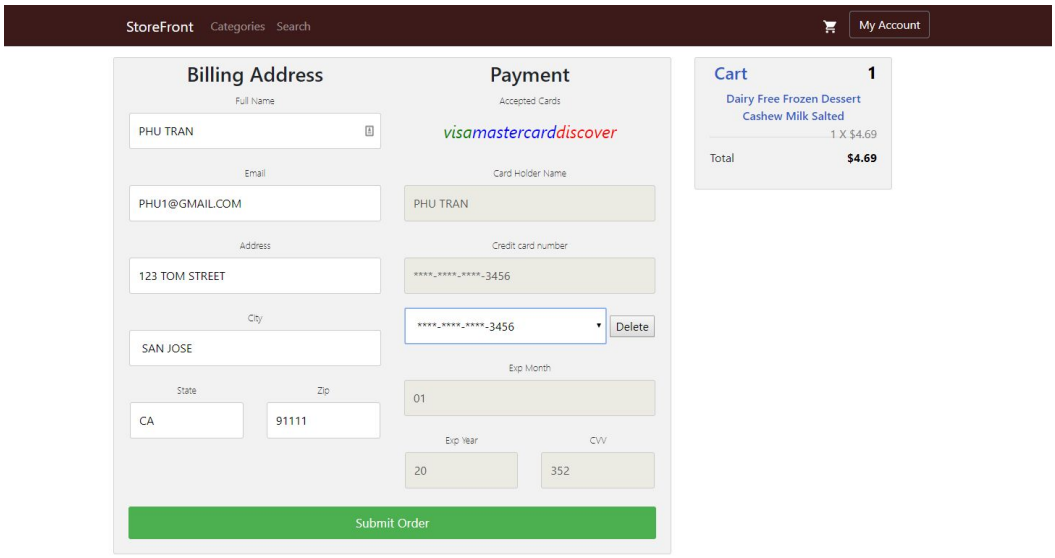


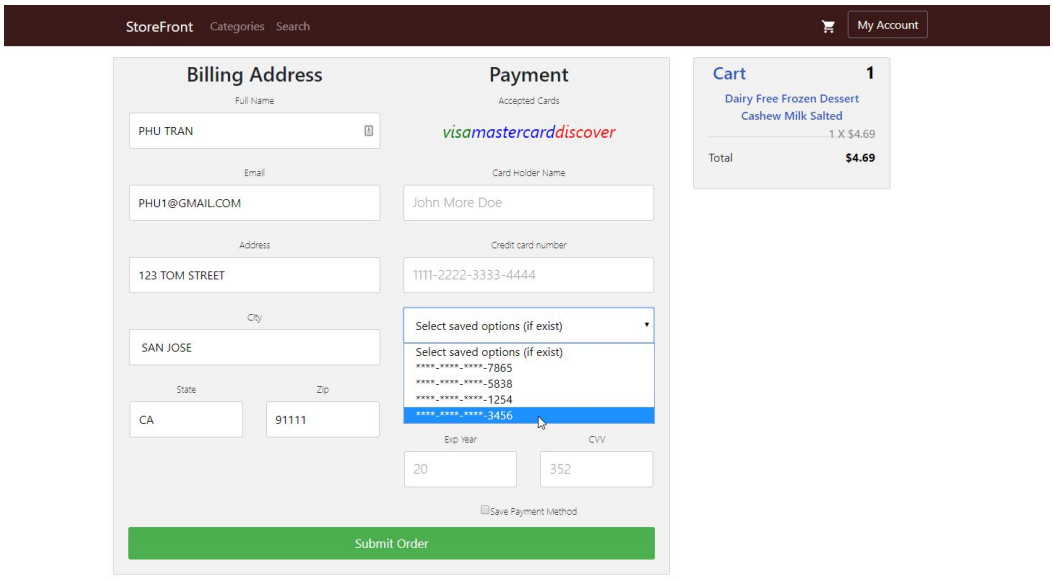


- View account information and email
    - Users will be able to view their username and other personal information such as date of birth through the application user interface.
    - The system will access the database to retrieve the data in the corresponding entity and display the information through the application UI.
    - Users can see their email displayed directly on the screen with a protected layer concealing some of the characters.
    - The system will access the database to retrieve the user's email and display it on the screen.

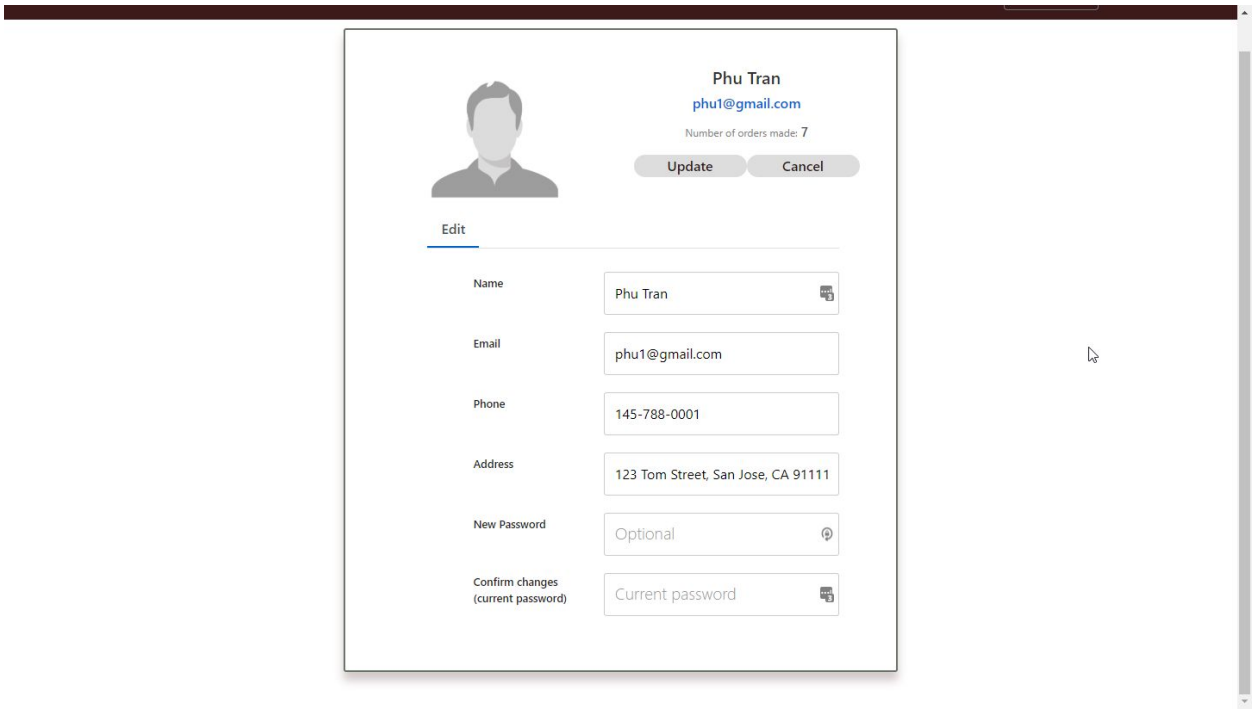


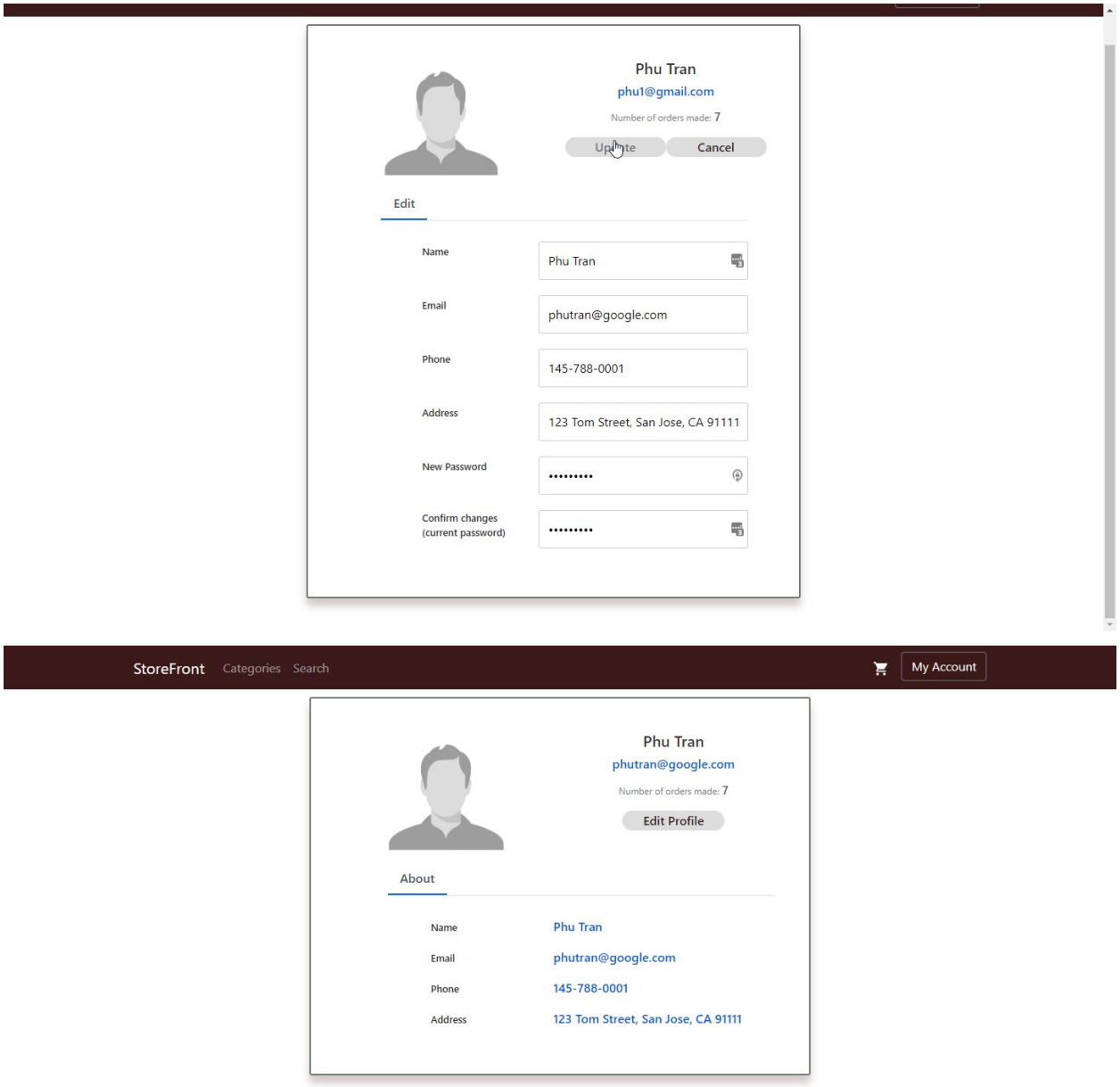
- View and selecting payment methods and information
    - Users can select already existing methods that they have created to check out their order faster.
    - The system will retrieve all payment options of the user and present it to them upon checking out items.
    - Users will be able to view all the credit card(s) that have saved in the system including: card number, card type, holder name, CVV, and card expiration date.
    - The system will access the database to retrieve the data in the corresponding entity and display the information through the application UI.



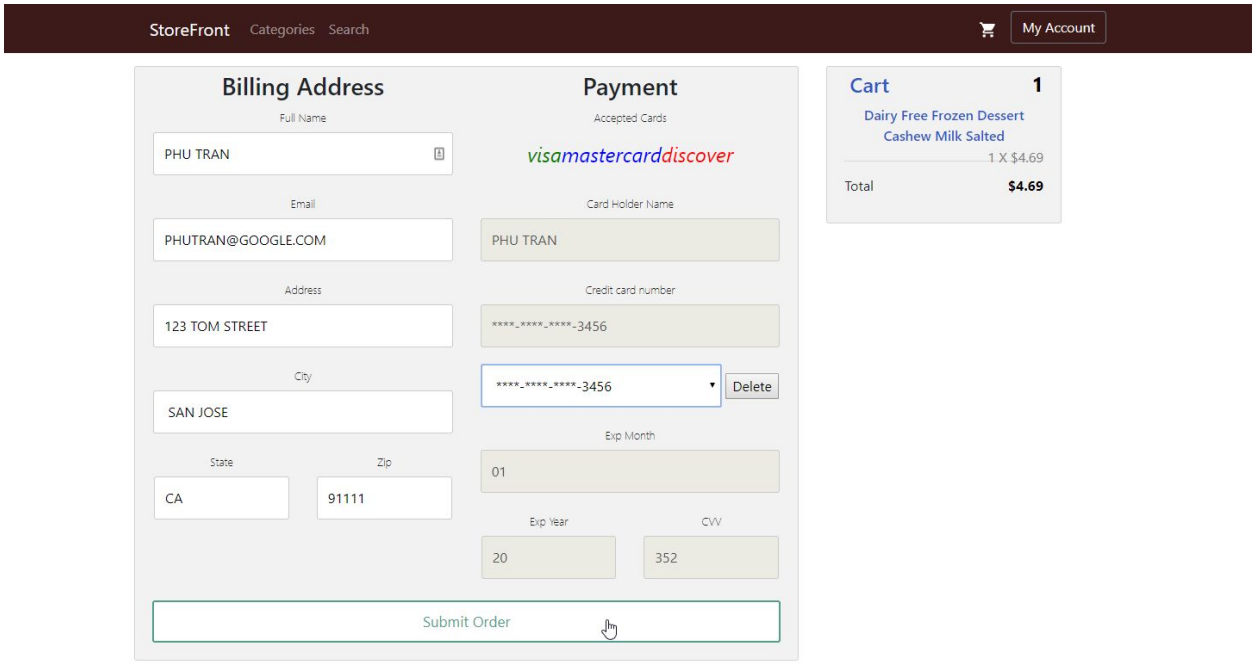


- Changing email and password
    - Users can change their email by confirming both an older email and a new email.
    - The system will check if the user provided the old email correctly with the database. If yes, replace the new email with the old one. Otherwise, keep the old email and display an error message through the screen.
    - Users can change their password by providing both an older password and a new password.
    - The system will check if the user provided the old password correctly with the database. If yes, replace the new password with the old one. Otherwise, keep the old password and display an error message through the screen.

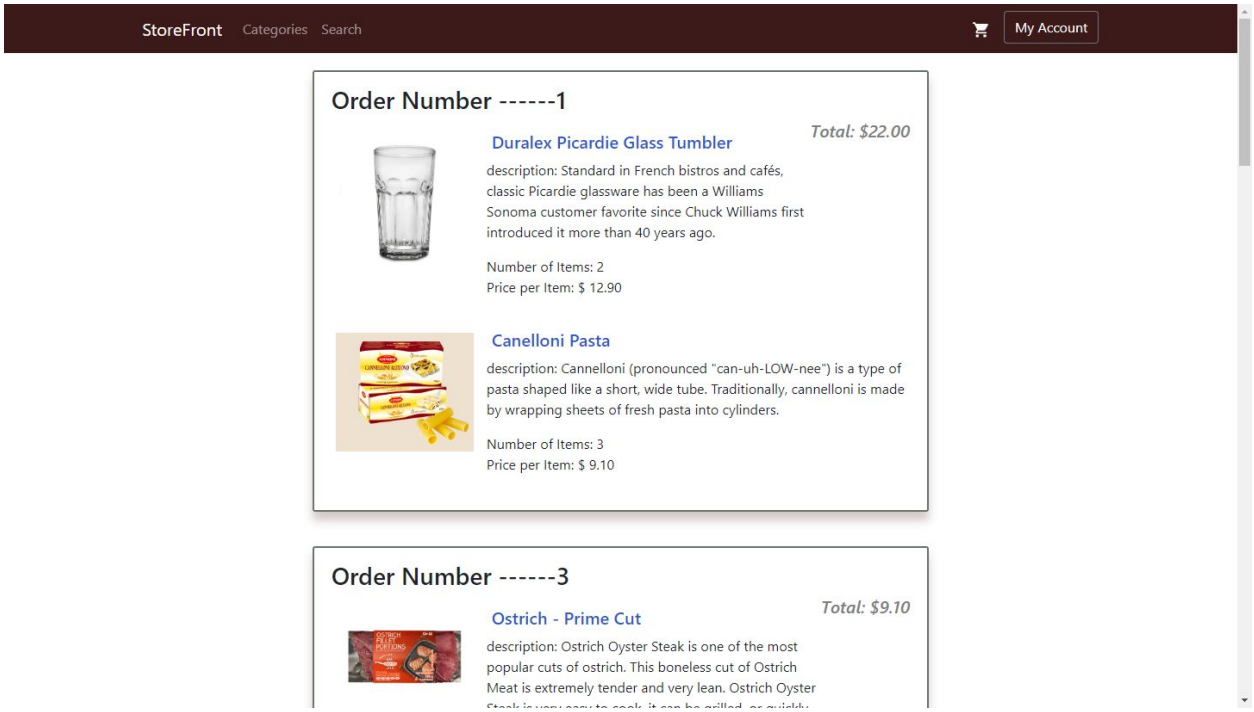



Phu Tran
phu1@gmail.com
Number of orders made: 7
Update
Cancel
Edit
Name
Phu Tran
Email
phutran@google.com
Phone
145-788-0001
Address
123 Tom Street, San Jose, CA 91111
New Password
Confirm changes
(current password)
StoreFront
Categories
Search
My Account
Phu Tran
phutran@google.com
Number of orders made: 7
Edit Profile
About
Name
Phu Tran
Email
phutran@google.com
Phone
145-788-0001
Address
123 Tom Street, San Jose, CA 91111

- Purchasing items
    - Users will be able to purchase their current cart of items using a new/existing payment method.
    - The system will display the cart's total price to the user.
    - The system will decrement the item(s) entity data to denote that there is less stock of said item.



- Viewing Order History
    - Users will be able to see orders that they made in the past.
    - The system will maintain a list of orders that the user made and return them on request.
    - Users will be able to view past orders as a list and click on an entry to view all details.



Non-functional Issues

- Detailed descriptions of Graphical User Interface

Our Graphical User Interface made use of HTML CSS and JavaScript to display content to users. To aide in making the user interface, we used React, a library that makes use of the three languages above to build scalable components.

The application's login page contains two fields, one for email and password. If the user has an existing account, they can enter their credentials in the associated fields and then click the Login Account button. If the user does not have an existing account, they can click the link below the Login Account button to navigate to the signup page. This page asks for a first name, last name, address, email and password from a user. After entering all fields, the user can then click the Create Account button and be redirected to the application's home page.

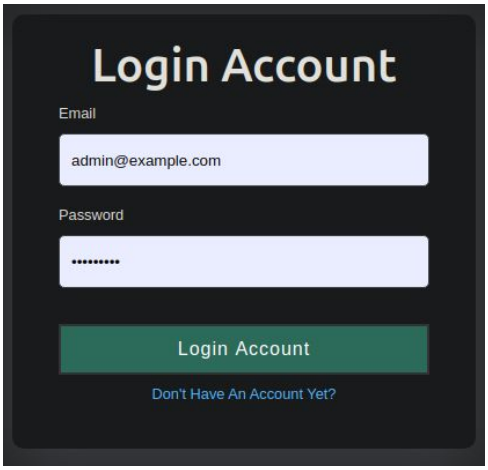
Present on all views of the application, we can find a navbar at the top of the page. The navbar has links to view categories, search for items, view the cart and the homepage. The navbar also has a dropdown titled "My Account." This dropdown holds information such as the user's email and links to the profile page, order history page and a link to log the user out. On the profile page, users can see all the information associated with their account such as their name, email, address and phone number. Upon clicking the "Edit Profile" button, users can change all of the aforementioned fields as well as set a new password, and confirm the changes by entering their current password.

For ordering items, users can find desired items on our website using one of three pages. The first page is the home page which is what the user first sees after logging in. Here, all items are displayed, and users can scroll through to see all tuples in the Item table. The second page to find items is that of the categories page, which displays all tuples of the Categories relation. Each entry in the categories page has a button that navigates the user to a new webpage. This webpage will display all items pertaining to the selected category. The third page to find items is the Search page. This page contains a search bar that will allow the user to search for items in the application either by category or item name. Upon entering the search term, all items that match the search query's criteria are returned. If the query yielded no results, the user will be notified that the query returned no results. In addition to the search page, the search bar can be found on all other pages so the user can search for items either by category or item name from anywhere in the application.

When purchasing items, users must navigate to the cart page. The cart page contains all of the items selected by the user and their associated quantities. The user can add to or remove from each item's order count. When the user wishes to purchase the items in their cart, the submit button at the bottom of the cart page is pressed, and they will be redirected to the payment page. Here, information such as billing address and card info is entered, with the cart's contents being visible to the right of the form for billing and payment. After submitting the information, the items will now be ordered and users can navigate to their order history to view their past purchases. The order history page displays all past orders made by the user. Each order has information such as the items that were bought and their total.

- Detailed descriptions of Security

Many measures have been taken to promote the security of the application. This includes hiding the characters in the fields where the user's password is entered. In addition to hiding password in the frontend of the application, the database does not store any passwords in plaintext. Instead, the hashes of each password is stored, to prevent sensitive data being exposed if our database was hacked or stolen.



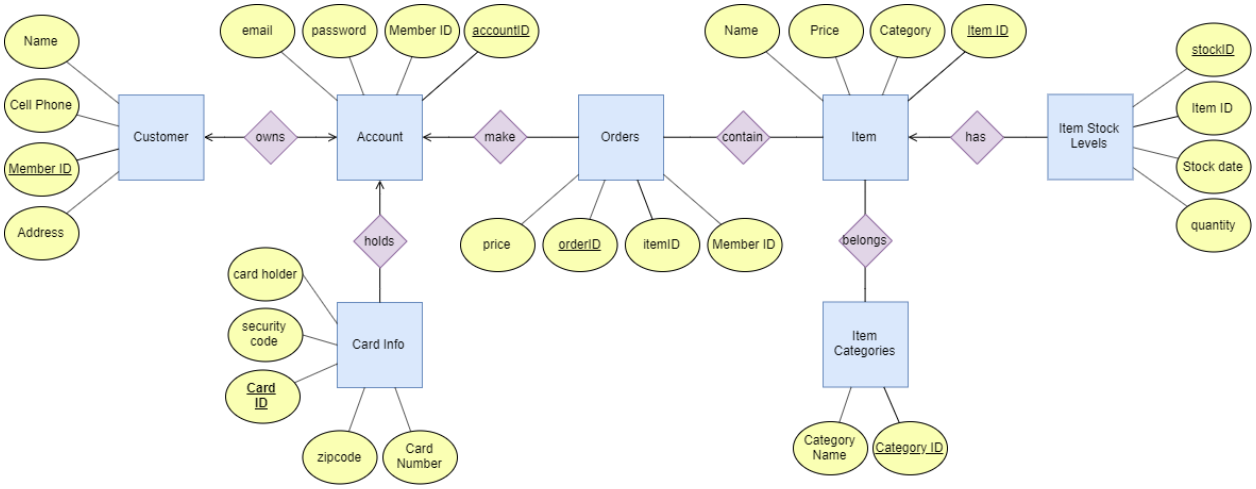
email	password
ascarsbrick0@blogspot.com	f651472f05ed929bab24b7da3a92d207
scrocket1@themeforest.net	21ec4b2edb2260da03aebb2a95fbcdca
rbasili2@alexa.com	3da13526855124534338fb5154668cc7
gpietruszewicz3@bbb.org	d6acc12d4d2bd2839fe6fc4b828165e
rbend4@webnode.com	bba8cf9d0c4b1f7b099b14cbd4bccf79

- Detailed descriptions of Access Control

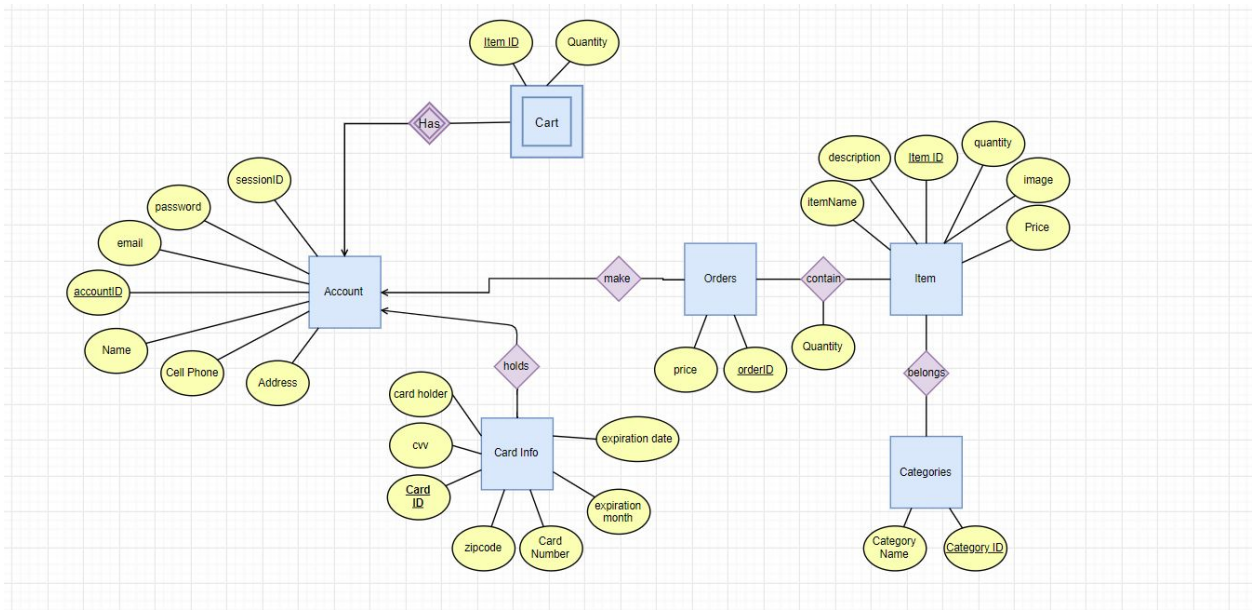
Signed out users cannot access any of the application's pages besides the login and sign up pages. To ensure this, before each webpage loads, the webpage ensures that a cookie exists with the user's accountID before loading. If the cookie isn't found, the application redirects the user from a route restricted to signed in users to the login page. When the user signs in, a cookie is stored on the browser, so the application can be used normally.

Project Design

Previous ERD Model of the application



Updated ERD Model for the application



Changes between previous and updated version

1. Remove the Customer entity set and move Customers' attributes to Account entity set instead

2. Add a sessionID for the Account entity set
3. Add a weak entity set named Cart that depended on Account entity
4. Add 2 more attributes (expired month and expired year) for the CardInfo entity set
5. Add 3 more attributes (description, quantity, and image) for the Item entity set
6. Fix Orders entity set to just contains only two attributes: price and orderID
7. Give the Contain relationship an extra attribute quantity
8. Remove stockLevel entity set

Updated ERD Model Description

Entity Sets and its explanations

- Account
    - This entity set contains the user's general information such as the user's name, address, phone number, password and account ID. The account ID is the primary key for the entity set.
    - It also holds the attribute, sessionID, which is used to determine whether the user is logged in or not (authenticated purpose).
    - This entity set shares 3 relationships with the other entity sets such as a one to many relationships with entity set Card Info, a one to many relationships with entity set Orders, and it also shared a weak entity relationship with the Cart entity set, where Account is the strong entity set while Cart is the weaker one.
- CardInfo
    - This entity set contains the user's card payment information where the primary key is the Card ID and the other attributes are card holder name, card number, card verification value, user’s zip code, card expired month and year.
    - This entity set shares a one to many relationships with the entity set Account.
- Orders
    - This entity set represents the user's shopping cart where the price is its attribute and the primary key will be its orderID.
    - This entity set shares a one to many relationships with the entity set Account
- Item

    - This entity holds the relevant information for each item which includes the item name, price, quantity (the amount of item the store still has), image, and description. The primary key for each entity is the item ID.
    - This entity set shares a many-to-many relationships with the item Category entity set.
- Categories
    - This is an entity set that will hold the various categories for Items. The primary key Category ID identifies entities in this set, Category Name is also another attribute. An example of Category name can be "Chips" or "Soda".
    - This entity set shares a many-to-many relationships with the Item entity set.

Weak Entity Sets and its explanations

- Cart
    - This is a weak entity set that depend on the Account entity to identify which item is currently in a cart of a specific account.

Relationship and dependencies explanations

- holds
    - This relationship has a one to many relationship between the entity sets Account and Card Info. It is a one to many relationship because each account can hold many different card information as a payment methods. On the other hand, we will assume that each card information can only be stored in exactly one account.
- make
    - This relationship has a one to many relationship between the entity sets Account and Orders. It is a one to many relationship because each account can make many orders and each specific order can be made by one account.
- contain
    - This relationship has a many-to-many relations between the entity sets Orders and Item. It is many-to-many because each order can hold many different items. The reason for this is because a person is not restricted to buying only a single item. On the same hand, one type of item can be ordered by many different order.
- belongs

- This is a relationship between Item and Item Categories. The relationship is many-to-many, as Items can have multiple categories and categories have multiple items.

List of complete non-trivial FDs

- Account's FD
  - accountID -> name, email, password, cell, address, sessionID
- Cart's FD
  - accountID, itemID -> quantity
- CardInfo's FD
  - cardID -> cardHolder, cardNumber, cvv, zipCode, expMonth, expYear
- Orders' FD
  - orderID -> price
- Item's FD
  - itemID -> itemName, price, quantity, image, description
- Categories' FD
  - categoryID -> categoryName
- Contain's FD
  - orderID, itemID -> quantity

As for make, belongs, and holds relationship table, it does not have any FD since all it contains is the keys

Normalization Process

Based on the list of non-trivial FD above and the revision of the ERD model, the tables are all in the form of BCNF. Hence we don't need to go to perform the normalization process.

Schemas and Database Models

Entity sets

- Account(accountID, email, password, name, cell, address, sessionID)

accountID	email	password	name	cell	address	sessionID
7	earnal6@quantcast.com	5963ff246cdb21b59d4f1a0f5ba3e000	Erna Arnal	983-528-1119	3009 La Follette Circle	7bg-8952-fbs87
8	lcharkham7@networksolutions.com	fd3145fe4c72570d0acf160004d95a1d	Liza Charkham	683-846-4023	50572 Cherokee Terrace	1357g8-mhj5-897
9	fmusslewhite8@timesonline.co.uk	8fb08a6d9197b4ed375116d6f6807f1a	Felipa Musslewhite	340-152-1797	8 Stephen Pass	a1d4-785b-794
10	ilemmon9@pcworld.com	736973eb31ddb430801d72e125eb1434	Innis Lemmon	991-223-7323	17 Toban Plaza	m1g4-85g-545-fh45
11	mtriplowa@google.co.jp	928ea5b041a58ec5810d8e24eab2887f	Mady Triplow	295-443-8330	8270 Green Crossing	5o45u-78o545-l2
12	rshadrackb@zimbio.com	82eaf1fb7d4dd80492393b5d035123c8	Reuben Shadrack	868-906-4924	5623 Victoria Drive	124f-452-k25-l45
13	fannettsc@fc2.com	a228ce69cb8a69afeb841507d238cdc3	Freeman Annetts	879-130-2731	921 Heath Pass	ht-45f-t4g-ff4
14	ikybertd@pagesperso-orange.fr	6abfa1dc3e8b0602486bea8b7731f9b5	Isac Kybert	370-449-1348	072 Luster Center	fg-784h-45fr-4b
15	tjerwoode@google.nl	de73719f0e26769f2cdaa99a8af89fe5	Teddie Jerwood	803-660-0104	199 Rowland Drive	87d-ht-454-bl4
16	jbinningf@naver.com	cee4937e8dd933d7a0ff6f7f0a1794a1	Justina Binning	151-469-1415	815 Dovetail Alley	1bl5-525-8f5dg-58...
17	lharesnapeg@shinystat.com	5d3e2927dd5ddddd521659cb7cbc3436	Lise Haresnape	925-304-2484	208 Bunker Hill Road	8411-787d-44f-45gd
18	nbyeh@ovh.net	99e9e9af8bc7619c747ed26f32289dde	Ninon Bye	521-582-2732	797 Eliot Drive	454eb-d554-f5d-r4...
19	cmonceyi@w3.org	c1bc8fb1d078142cffaa0e9818b6aa6a	Charles Moncey	943-403-6754	56 Mccormick Terrace	ef451-895s-45fs-5...
20	aastlej@mysql.com	81b4d83856aa1beff8eeb34bb23bd841	Adoree Astle	406-383-4369	2 Katie Plaza	215e-sv54-s5852

- CardInfo(cardID, cardHolder, cardNumber, cvv, zipCode, expMonth, expYear)

cardID	CardHolder	CVV	Zip	CardNumber	ExpMonth	ExpYear
1	Samuel Roger	845	974512	4547-2215-5589-7865	02	24
2	Thomas Austin	784	98541	5214-2015-5632-4512	05	22
3	Keith Jam	124	95115	3564-1245-7841-8485	04	20
4	Gena Hopewell	168	93213	3539-1283-2029-5838	12	22
5	Bob Dang	985	98511	7454-1245-5457-6932	08	25
6	Cinderella McRitchie	212	93202	5002-3566-6226-9946	12	22
7	Hildagard Symington	651	95075	5610-4590-1040-5420	12	21
8	Eddy Ung	451	96235	7452-1254-4755-6542	01	27
9	Chiquia Dennington	643	91256	5183-0293-8061-3577	09	24
10	Kalindi Schubart	559	95653	3565-2575-2781-0847	07	23
11	Ronny Sa	541	97512	4564-2535-5667-2633	05	24
12	Odell McNaught	692	94068	3536-4509-0939-8434	01	21
13	Michele Sawl	623	92195	3567-3354-4948-4301	01	25
14	Aili Rean	273	91062	4211-9517-2088-6036	11	23
15	Heddi Esposita	518	91799	2018-8097-9437-1003	11	26
16	Loreen McIndoe	685	94621	3580-1560-6271-4789	12	24
17	Clea Allright	778	94068	2723-0115-9528-8936	06	22
18	Isaac Marlowe	685	92035	3038-1288-3039-8734	07	25
19	Alica Legh	303	94910	5038-9081-6431-0355	02	24
20	Tom Cruz	456	95132	4575-8956-4521-1254	11	27

- Orders(orderID, price)

orderID	price
1	53.1
2	18.2
3	9.1
4	6.3
5	22.5
6	23.99
7	6.2
8	9.9
9	7.1
10	75.5
11	15.1
12	4.75
13	15.3
14	102.16
15	49.99

- Item(itemID, itemName, price, quantity, image, description)

itemID	itemName	price	description	quantity	image
1	Duralex Picardie Glass Tumbler	12.9	Standard in French bistros and cafés, classic Pic...	20	glass_item.png
2	Canelloni Pasta	9.1	Cannelloni (pronounced "can-uh-LOW-nee") is a...	50	pasta_item.png
3	Ostrich - Prime Cut	9.1	Ostrich Oyster Steak is one of the most popular...	31	ostrich_item.png
4	Turkey - Ground. Lean	6.3	JENNIE-O® All Natural* Lean Ground Turkey is ...	73	turkey_item.png
5	Doilies Paper	4.5	These white paper doilies are a classic way to s...	50	doilies_item.png
6	Chantrelle Mushroom	23.99	The Chanterelle is a distinctively trumpet-shape...	100	mushroom_item.png
7	Foam Dinner Plate	3.1	White color and great for hot and cold food.	150	plate_item.png
8	Pasta - Rotini, Dry	3.3	Barilla Rotini is Non-GMO verified, 100% durum ...	50	pasta2_item.png
9	Longos - Chicken Curried	7.1	A India inspired soup with chunks of tender chic...	120	curry_item.png
10	Lettuce - Iceberg	2.09	Iceberg is a variety of lettuce with crisp leaves ...	50	letture_item.png
11	Wine - Cotes Du Rhone	15.1	Medium garnet color, with bright red reflections...	160	wine_item.png
12	Garbage Bag - Clear	4.75	The only tall kitchen recycling bags with odor co...	50	trashbag_item.png
13	Pumpkin	5.1	A pumpkin is a cultivar of a squash plant, most c...	50	pumpkin_item.png
14	Piping Jelly - All Colours	25.54	Highly concentrated vibrant colors mix easily an...	40	icying_item.png
15	Wine - Zonnebloem Pinotage	49.99	A complex wine with red fruit flavors. Big, elega...	50	wine2_item.png
16	Cheez-It Baked Snack Crackers	4.67	Baked Snack Crackers, Original Cheddar, Family...	81	cheezIt.png
17	22CT GRAB & SNACK	10	Bring your favorite Wise snacks with you where...	18	variety_packs.png
18	Pringles Potato Crisps Original...	1.54	Satisfy your snack craving with the salty, stack...	62	potato-chips.png
19	Breyers Frozen Dairy Dessert	4.49	Oreo cookie pieces in vanilla flavored frozen dai...	33	breyers-ice-cream.png
20	Dairy Free Frozen Dessert Ca...	4.69	Rich, creamy & totally indulgent! Certified glute...	21	cashew-milk-ice-cream.png
21	Mountain Dew Soda	26.57	Pack of 24, 12 ounces.cans. 170 calories per ca...	35	mountain-dew-soda.png
22	Zevia Zero Calorie Soda, Rain...	22.14	MORE THAN A BEVERAGE: Zevia is a movement...	42	zevia-soda.png
23	Fanta Orange Soda Fruit Flav...	6.66	Pack of twelve, 12 FL OZ per can of naturally fl...	32	fanta-soda.png
24	DOLE Sliced Yellow Cling Peac...	2.88	HEALTHY SNACKS AND JUICES: From packaged...	23	dole-peaches.png
25	Jell-O No Bake Cherry Chees...	3.14	One 17.8 oz. package of Jell-O No Bake Cherry ...	34	chesse-cake.png
26	Quest Nutrition Protein Bar, B...	0.99	Craveable Birthday Cake flavored coated protei...	100	cake-bar.png
27	Nescafe Taster's Choice Insta...	2.99	Made for an 8 ounce Cup, each of the 16 conve...	99	instant-coffee.png
28	Maruchan Ramen Noodles Chi...	3.5	Maruchan Souper 6 Pack Ramen Noodle Soup C...	88	instant-noodle.png
29	McCormick Panko Bread Crum...	7.87	McCormick Panko Bread Crumbs Italian Herb, 21...	34	bread-crumbs.png
30	Krusteaz No Knead Hawaiian ...	2.45	Delightful Hawaiian Sweet Bread mix, just witho...	21	bread-mix.png

- Categories(categoryID, categoryName)

categoryID	categoryName
1	Snacks
2	Ice Cream
3	Sweet
4	Soda
5	Fruit
6	Veggies
7	Wine
8	Meat
9	Pasta
10	Mushroom
11	Kitchen Utensils
12	Instant Food
13	Cake
14	Bread
15	Processed Food

- Cart(accountID, itemID, quantity)

accountID	itemID	quantity
1	1	1
1	2	1
1	4	3
2	1	1
5	1	2
5	6	1
6	7	3
6	9	1
6	10	1
7	8	1
7	9	2
8	1	1
8	9	2
10	2	1
10	3	2
10	5	1
10	6	1

Relationships

- holds(accountID, cardID)

accountID	cardID
2	4
3	5
5	6
6	7
7	8
7	9
7	10
11	11
12	12
13	13
14	15
14	16
14	17
16	14
18	18
19	19
20	20

- make(accountID, orderID)

accountID	orderID
1	1
1	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15

- contain(orderID, itemID, quantity)

orderID	itemID	quantity
1	1	2
1	2	3
2	2	2
3	3	1
4	4	1
5	5	4
6	6	1
7	7	2
8	8	3
9	9	1
10	11	5
11	11	1
12	12	1
13	13	3
14	14	4
15	15	1

- belongs(itemID, categoryID)

itemID	categoryID
1	11
2	9
3	8
4	8
5	11
6	6
6	10
7	11
8	9
9	12
9	15
10	6
11	7
12	11
13	5
14	3
14	15
15	7
16	1
16	15
17	1
17	15
18	1
18	15

19	2
19	3
20	2
20	3
21	3
21	4
21	15
22	3
22	4
22	15
23	3
23	4
23	15
24	5
24	15
25	3
25	13
25	15

26	1
26	3
26	13
26	15
27	12
28	12
28	15
29	14
30	14
30	15

(Note: the relationships of weak entity set does not being converted)

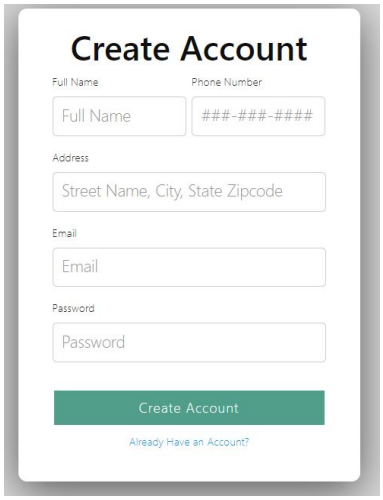
Implementation

Our implementation focused on designing around the functional requirements that we had set from the start. This led to the following code designs for the front and back end in each section.

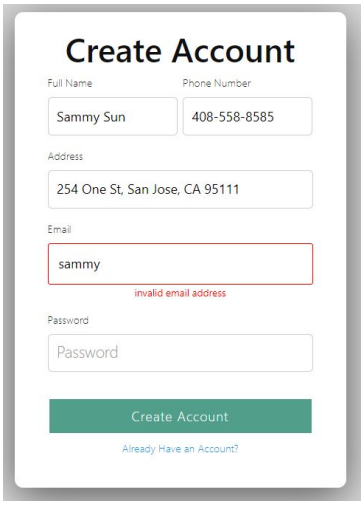
Creating an Account

User Interface after fill out the registration form and passes all the check such as correct phone format, the user account is inserted to the Account table

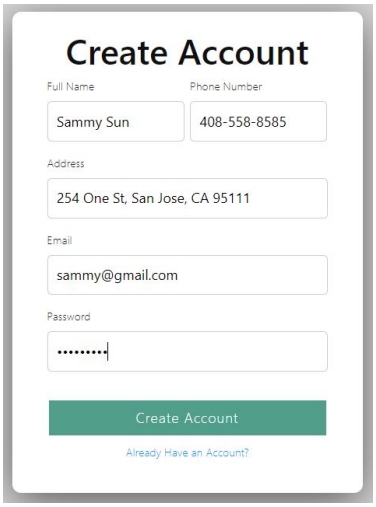
Before



During filling process



After finish



Database, before the “Create Account” button is being clicked:

```
5 •  SELECT * FROM Storefront.Account;
```

esult Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

accountID	email	password	name	cell	address	sessionID
1	ascarsbrick0@blogspot.com	f651472f05ed929bab24b7da3a92d207	Albina Scarsbrick	273-816-3890	08106 Golf View Street	145fr-545-f4r-45
2	scrocket1@themeforest.net	21ec4b2edb2260da03aebb2a95fbcdlca	Sam Crocket	202-468-7307	7752 Pine View Circle	a14-7845-f5d-554r
3	nbasili2@alexa.com	3da13526855124534338fb5154668cc7	Nolie Basili	994-128-9298	334 Melody Plaza	54bo-45li5-85
4	gpietruszewicz3@bbb.org	d6acc12d4d2bd2839fe6fc4b828165e	Gail Pietruszewic	269-647-6574	7 Green Ridge Street, Santa Clara, CA 95842	jk-7841-hy55
5	rbend4@webnode.com	bba8cf9d0c4b1f7b099b14cbd4bccf79	Row Bend	747-339-2047	727 Dunning Court	125g-jk2-552
6	tpoundford5@wikispaces.com	eb7140ec45b0jca2be9a8c4a89f98197d	Terrel Poundford	708-331-7934	829 Rowland Trail	m12g5-452g
7	earnal6@quantcast.com	5963ff246cdb21b59d4f1a0f5ba3e000	Erna Arnal	983-528-1119	3009 La Follette Circle	7bg-8952-fbs87
8	lcharkham7@networksolutions.com	fd3145fe4c72570d0acf160004d95a1d	Liza Charkham	683-846-4023	50572 Cherokee Terrace	1357g8-mhj5-897
9	fmusslewhite8@timesonline.co.uk	8fb08a6d9197b4ed375116d6f6807f1a	Felipa Musslewhite	340-152-1797	8 Stephen Pass	a1d4-785b-794
10	ilemmon9@pcworld.com	736973eb31ddb430801d72e125eb1434	Innis Lemmon	991-223-7323	17 Toban Plaza	m1g4-85g-545-fh45
11	mtriplowa@google.co.jp	928ea5b041a58ec5810d8e24eab2887f	Mady Triplow	295-443-8330	8270 Green Crossing	5o45u-78o545-l2
12	rshadrackb@zimbio.com	82eaf1fb7d4dd80492393b5d035123c8	Reuben Shadrack	868-906-4924	5623 Victoria Drive	124f-452-k25-l45
13	fannettsc@fc2.com	a228ce69cb8a69afeb841507d238cdc3	Freeman Annetts	879-130-2731	921 Heath Pass	ht-45f-t4g-ff4
14	ikybertd@pagesperso-orange.fr	6abfa1dc3e8b0602486bea8b7731f9b5	Isac Kybert	370-449-1348	072 Luster Center	fg-784h-45fr-4b
15	tjerwoode@google.nl	de73719f0e26769f2cdaa99a8af89fe5	Teddie Jerwood	803-660-0104	199 Rowland Drive	87d-ht-454-bl4
16	jbinningf@naver.com	cee4937e8dd933d7a0ff6f7f0a1794a1	Justina Binning	151-469-1415	815 Dovetail Alley	1bl5-525-8f5dg-58...
17	lharesnapeg@shinystat.com	5d3e2927dd5ddddd521659cb7cbc3436	Lise Haresnape	925-304-2484	208 Bunker Hill Road	8411-787d-44f-45gd
18	nbyeh@ovh.net	99e9e9af8bc7619c747ed26f32289dde	Ninon Bye	521-582-2732	797 Eliot Drive	454eb-d554-f5d-r4...
19	cmonceyi@w3.org	c1bc8fb1d078142cffaa0e9818b6aa6a	Charles Moncey	943-403-6754	56 Mccormick Terrace	ef451-895s-45fs-5...
20	aastlej@mysql.com	81b4d83856aa1beff8eeb34bb23bd841	Adoree Astle	406-383-4369	2 Katie Plaza	215e-sv54-s5852
NULL	NULL	NULL	NULL	NULL	NULL	NULL

After the “Create Account” button is being clicked:

```
5 •  SELECT * FROM Storefront.Account;
```

esult Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

accountID	email	password	name	cell	address	sessionID
1	ascarsbrick0@blogspot.com	f651472f05ed929bab24b7da3a92d207	Albina Scarsbrick	273-816-3890	08106 Golf View Street	145fr-545-f4r-45
2	scrocket1@themeforest.net	21ec4b2edb2260da03aebb2a95fbcdca	Sam Crocket	202-468-7307	7752 Pine View Circle	a14-7845-f5d-554r
3	nbasili2@alexa.com	3da13526855124534338fb5154668cc7	Nolie Basili	994-128-9298	334 Melody Plaza	54bo-45li5-85
4	gpietruszewicz3@bbb.org	d6acc12d4d2bd2839fe6fc4b828165e	Gail Pietruszewic	269-647-6574	7 Green Ridge Street, Santa Clara, CA 95842	jk-7841-hy55
5	rbend4@webnode.com	bba8cf9d0c4b1f7b099b14cbd4bccf79	Row Bend	747-339-2047	727 Dunning Court	125g-jk2-552
6	tpoundford5@wikispaces.com	eb7140ec45b0jca2be9a8c4a89f98197d	Terrel Poundford	708-331-7934	829 Rowland Trail	m12g5-452g
7	earnal6@quantcast.com	5963ff246cdb21b59d4f1a0f5ba3e000	Erna Arnal	983-528-1119	3009 La Follette Circle	7bg-8952-fbs87
8	lcharkham7@networksolutions.com	fd3145fe4c72570d0acf160004d95a1d	Liza Charkham	683-846-4023	50572 Cherokee Terrace	1357g8-mhj5-897
9	fmusslewhite8@timesonline.co.uk	8fb08a6d9197b4ed375116d6f6807f1a	Felipa Musslewhite	340-152-1797	8 Stephen Pass	a1d4-785b-794
10	ilemmon9@pcworld.com	736973eb31ddb430801d72e125eb1434	Innis Lemmon	991-223-7323	17 Toban Plaza	m1g4-85g-545-fh45
11	mtriplowa@google.co.jp	928ea5b041a58ec5810d8e24eab2887f	Mady Triplow	295-443-8330	8270 Green Crossing	5o45u-78o545-l2
12	rshadrackb@zimbio.com	82eaf1fb7d4dd80492393b5d035123c8	Reuben Shadrack	868-906-4924	5623 Victoria Drive	124f-452-k25-l45
13	fannettsc@fc2.com	a228ce69cb8a69afeb841507d238cdc3	Freeman Annetts	879-130-2731	921 Heath Pass	ht-45f-t4g-ff4
14	ikybertd@pagesperso-orange.fr	6abfa1dc3e8b0602486bea8b7731f9b5	Isac Kybert	370-449-1348	072 Luster Center	fg-784h-45fr-4b
15	tjerwoode@google.nl	de73719f0e26769f2cdaa99a8af89fe5	Teddie Jerwood	803-660-0104	199 Rowland Drive	87d-ht-454-bl4
16	jbinningf@naver.com	cee4937e8dd933d7a0ff6f7f0a1794a1	Justina Binning	151-469-1415	815 Dovetail Alley	1bl5-525-8f5dg-58...
17	lharesnapeg@shinystat.com	5d3e2927dd5ddddd521659cb7cbc3436	Lise Haresnape	925-304-2484	208 Bunker Hill Road	8411-787d-44f-45gd
18	nbyeh@ovh.net	99e9e9af8bc7619c747ed26f32289dde	Ninon Bye	521-582-2732	797 Eliot Drive	454eb-d554-f5d-r4...
19	cmonceyi@w3.org	c1bc8fb1d078142cffaa0e9818b6aa6a	Charles Moncey	943-403-6754	56 Mccormick Terrace	ef451-895s-45fs-5...
20	aastlej@mysql.com	81b4d83856aa1beff8eeb34bb23bd841	Adoree Astle	406-383-4369	2 Katie Plaza	215e-sv54-s5852
21	sammy@gmail.com	fab459fa5a45eb2402f7cef48ca630b8	Sammy Sun	408-558-8585	254 One St, San Jose, CA 95111	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Code explanation

- Frontend

```
export async function registerUser(newUser) {
  let objects = [];
  await fetch(`http://${url}/users/add?email=${newUser.email}` +
    `&password=${newUser.password}&name='${newUser.name}'&` +
    `cell=${newUser.cell}&address='${newUser.address}'`)
    .then((response) => response.json())
    .then((response) => {
      objects = response;
    })
    .catch((err) => {
      console.error(err);
    });
  return objects;
}
```

- Backend

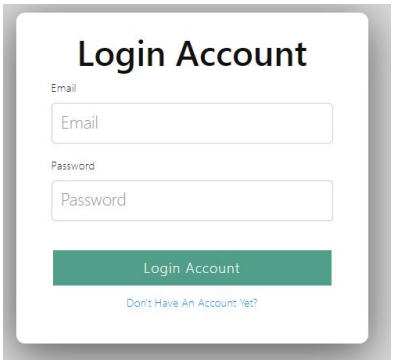
```
app.get('/users/add', (req, res) => {
    const { email, password, name, cell, address } = req.query;
    pool.getConnection(function (err, con) {
        var hashedPassword = md5(password);
        con.query(`select accountID from Account where email='${email}'`, (err, results) => {
            if (err) res.send(err);
            else {
                if (results.length == 0) {
                    con.query(`insert into Account (email, password, name, cell, address) values(
                        '${email.trim()}',
                        '${hashedPassword}',
                        '${name}',
                        '${cell}',
                        '${address}'
                        )`, (err, results) => {
                        if (err) res.send(err);
                        else res.send({
                            accountID: results.insertId
                        });
                    });
                } else res.send({
                    accountID: "Email already exists"
                });
            }
        });
        con.release();
    });
});
```

  - First the function will select the newly created ID assigned to the account and then it will assign the values that are passed into the tuple.

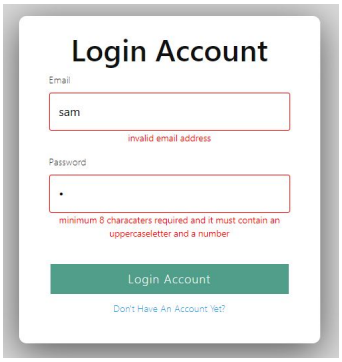
Login

User Interface the user will be lead to this page will they just created an account in the register page for the relogin after their session is expired. What this page does is getting the email and password from the user's input to check with the database to authenticated whether the user is who they say they are. It also will assign and store a new session ID inside the Account table.

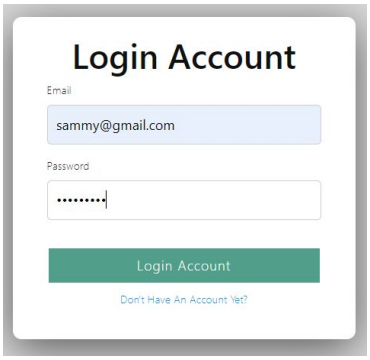
Before



During filling process



After finish



Database, when the “Login Account” button is being clicked it will just get the data from database and store a new session ID inside the Account table:

```
5 •  SELECT * FROM Storefront.Account;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

accountID	email	password	name	cell	address	sessionID
1	ascarsbrick0@blogspot.com	f651472f05ed929bab24b7da3a92d207	Albina Scarsbrick	273-816-3890	08106 Golf View Street	145fr-545-f4r-45
2	scrocket1@themeforest.net	21ec4b2edb2260da03aebb2a95fbcdlca	Sam Crocket	202-468-7307	7752 Pine View Circle	a14-7845-f5d-554r
3	nbasili2@alexa.com	3da13526855124534338fb5154668cc7	Nolie Basili	994-128-9298	334 Melody Plaza	54bo-45li5-85
4	gpietruszewicz3@bbb.org	d6acc12d4d2bd2839fe6fc4b828165e	Gail Pietruszewic	269-647-6574	7 Green Ridge Street, Santa Clara, CA 95842	jk-7841-hy55
5	rbend4@webnode.com	bba8cf9d0c4b1f7b099b14cbd4bccf79	Row Bend	747-339-2047	727 Dunning Court	125g-jk2-552
6	tpoundford5@wikispaces.com	eb7140ec45b0jca2be9a8c4a89f98197d	Terrel Poundford	708-331-7934	829 Rowland Trail	m12g5-452g
7	earnal6@quantcast.com	5963ff246cdb21b59d4f1a0f5ba3e000	Erna Arnal	983-528-1119	3009 La Follette Circle	7bg-8952-fbs87
8	lcharkham7@networksolutions.com	fd3145fe4c72570d0acf160004d95a1d	Liza Charkham	683-846-4023	50572 Cherokee Terrace	1357g8-mhj5-897
9	fmusslewhite8@timesonline.co.uk	8fb08a6d9197b4ed375116d6f6807f1a	Felipa Musslewhite	340-152-1797	8 Stephen Pass	a1d4-785b-794
10	ilemmon9@pcworld.com	736973eb31ddb430801d72e125eb1434	Innis Lemmon	991-223-7323	17 Toban Plaza	m1g4-85g-545-fh45
11	mtriplowa@google.co.jp	928ea5b041a58ec5810d8e24eab2887f	Mady Triplow	295-443-8330	8270 Green Crossing	5o45u-78o545-l2
12	rshadrackb@zimbio.com	82eaf1fb7d4dd80492393b5d035123c8	Reuben Shadrack	868-906-4924	5623 Victoria Drive	124f-452-k25-l45
13	fannettsc@fc2.com	a228ce69cb8a69afeb841507d238cdc3	Freeman Annetts	879-130-2731	921 Heath Pass	ht-45f-t4g-ff4
14	ikybertd@pagesperso-orange.fr	6abfa1dc3e8b0602486bea8b7731f9b5	Isac Kybert	370-449-1348	072 Luster Center	fg-784h-45fr-4b
15	tjerwoode@google.nl	de73719f0e26769f2cdaa99a8af89fe5	Teddie Jerwood	803-660-0104	199 Rowland Drive	87d-ht-454-bl4
16	jbinningf@naver.com	cee4937e8dd933d7a0ff6f7f0a1794a1	Justina Binning	151-469-1415	815 Dovetail Alley	1bl5-525-8f5dg-58...
17	lharesnapeg@shinystat.com	5d3e2927dd5ddddd521659cb7cbc3436	Lise Haresnape	925-304-2484	208 Bunker Hill Road	8411-787d-44f-45gd
18	nbyeh@ovh.net	99e9e9af8bc7619c747ed26f32289dde	Ninon Bye	521-582-2732	797 Eliot Drive	454eb-d554-f5d-r4...
19	cmonceyi@w3.org	c1bc8fb1d078142cffaa0e9818b6aa6a	Charles Moncey	943-403-6754	56 Mccormick Terrace	ef451-895s-45fs-5...
20	aastlej@mysql.com	81b4d83856aa1beff8eeb34bb23bd841	Adoree Astle	406-383-4369	2 Katie Plaza	215e-sv54-s5852
21	sammy@gmail.com	fab459fa5a45eb2402f7cef48ca630b8	Sammy Sun	408-558-8585	254 One St, San Jose, CA 95111	3e5bf0f1-f08b-5ed...
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Code explanation

- Frontend

```
export async function getUser(user) {
  let objects = [];
  await fetch(`http://${url}/users?id=${user.accountID}`)
    .then((response) => response.json())
    .then((response) => {
      objects = response;
    })
    .catch((err) => {
      console.error(err);
    });
  return objects;
}
```

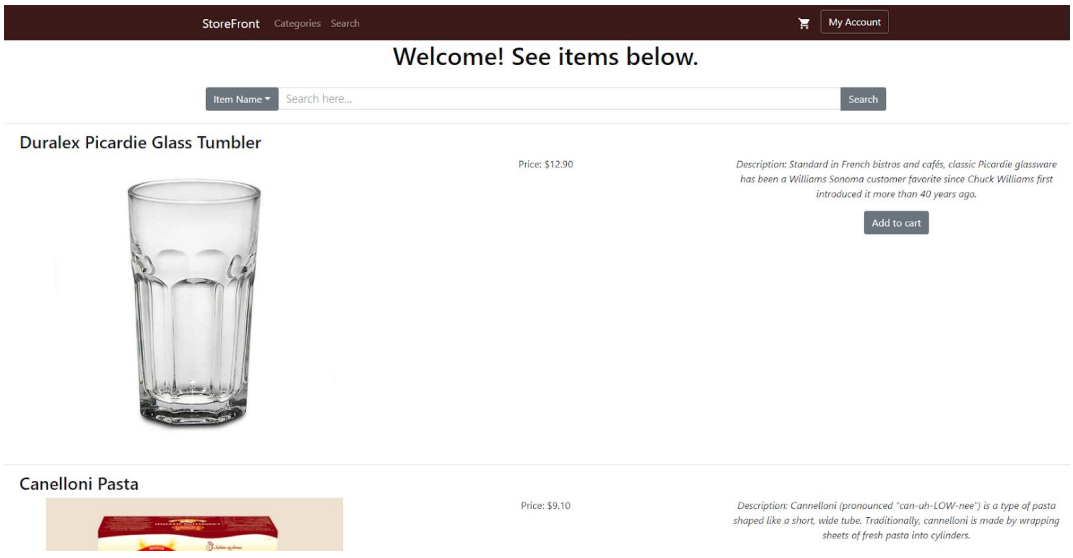
- Backend

```
app.get('/users', (req, res) => {
    const { id } = req.query;
    pool.getConnection(function (err, con) {
        con.query(`select * from Account where accountID=${id}`, (err, results) => {
            if (err) res.send(err);
            else {
                res.send({
                    ...results
                });
            }
        });
        con.release();
    });
});
```

  - The function simply returns the tuple where the sent accountID is found since accountID is the key.

Browse Items

User Interface: After the user logged in, the first thing they see is that they can see all the items that are inside the store.



Add to cart

Ostrich - Prime Cut

Price: $9.10

Description: Ostrich Oyster Steak is one of the most popular cuts of ostrich. This boneless cut of Ostrich Meat is extremely tender and very lean. Ostrich Oyster Steak is very easy to cook, it can be grilled, or quickly pan fried. The Oyster Steak is wonderful just cooked on its own to appreciate the full natural flavor. The great thing about Ostrich Oyster Steak is that it can be cooked to your taste, and can even be eaten raw (as Carpaccio) so a little pink in the middle is ideal and will maximize your enjoyment of this product.

Add to cart

Turkey - Ground. Lean

Price: $6.30

Description: JENNIE-O® All Natural Lean Ground Turkey is packed with nutrition, making it an amazing alternative to ground beef. Create delicious versions of your family's favorite recipes, like tacos, meatballs, casseroles,*

Database, It will get all the item inside the database by itemID:

itemID	itemName	price	description	quantity	image
1	Duralex Picardie Glass Tumbler	12.9	Standard in French bistros and cafés, classic Pic...	20	glass_item.png
2	Canelloni Pasta	9.1	Cannelloni (pronounced "can-uh-LOW-nee") is a...	50	pasta_item.png
3	Ostrich - Prime Cut	9.1	Ostrich Oyster Steak is one of the most popular...	31	ostrich_item.png
4	Turkey - Ground. Lean	6.3	JENNIE-O® All Natural* Lean Ground Turkey is ...	73	turkey_item.png
5	Doilies Paper	4.5	These white paper doilies are a classic way to s...	50	doilies_item.png
6	Chantrelle Mushroom	23.99	The Chanterelle is a distinctively trumpet-shape...	100	mushroom_item.png
7	Foam Dinner Plate	3.1	White color and great for hot and cold food.	150	plate_item.png
8	Pasta - Rotini, Dry	3.3	Barilla Rotini is Non-GMO verified, 100% durum ...	50	pasta2_item.png
9	Longos - Chicken Curried	7.1	A India inspired soup with chunks of tender chic...	120	curry_item.png
10	Lettuce - Iceberg	2.09	Iceberg is a variety of lettuce with crisp leaves ...	50	letture_item.png
11	Wine - Cotes Du Rhone	15.1	Medium garnet color, with bright red reflections...	160	wine_item.png
12	Garbage Bag - Clear	4.75	The only tall kitchen recycling bags with odor co...	50	trashbag_item.png
13	Pumpkin	5.1	A pumpkin is a cultivar of a squash plant, most c...	50	pumpkin_item.png
14	Piping Jelly - All Colours	25.54	Highly concentrated vibrant colors mix easily an...	40	icying_item.png
15	Wine - Zonnebloem Pinotage	49.99	A complex wine with red fruit flavors. Big, elega...	50	wine2_item.png
16	Cheez-It Baked Snack Crackers	4.67	Baked Snack Crackers, Original Cheddar, Family...	81	cheezIt.png
17	22CT GRAB & SNACK	10	Bring your favorite Wise snacks with you where...	18	variety_packs.png
18	Pringles Potato Crisps Original...	1.54	Satisfy your snack craving with the salty, stack...	62	potato-chips.png
19	Breyers Frozen Dairy Dessert	4.49	Oreo cookie pieces in vanilla flavored frozen dai...	33	breyers-ice-cream.png
20	Dairy Free Frozen Dessert Ca...	4.69	Rich, creamy & totally indulgent! Certified glute...	21	cashew-milk-ice-cream.png
21	Mountain Dew Soda	26.57	Pack of 24, 12 ounces.cans. 170 calories per ca...	35	mountain-dew-soda.png
22	Zevia Zero Calorie Soda, Rain...	22.14	MORE THAN A BEVERAGE: Zevia is a movement...	42	zevia-soda.png
23	Fanta Orange Soda Fruit Flav...	6.66	Pack of twelve, 12 FL OZ per can of naturally fl...	32	fanta-soda.png
24	DOLE Sliced Yellow Cling Peac...	2.88	HEALTHY SNACKS AND JUICES: From packaged...	23	dole-peaches.png
25	Jell-O No Bake Cherry Chees...	3.14	One 17.8 oz. package of Jell-O No Bake Cherry ...	34	chesse-cake.png
26	Quest Nutrition Protein Bar, B...	0.99	Craveable Birthday Cake flavored coated protei...	100	cake-bar.png
27	Nescafe Taster's Choice Insta...	2.99	Made for an 8 ounce Cup, each of the 16 conve...	99	instant-coffee.png
28	Maruchan Ramen Noodles Chi...	3.5	Maruchan Souper 6 Pack Ramen Noodle Soup C...	88	instant-noodle.png
29	McCormick Panko Bread Crum...	7.87	McCormick Panko Bread Crumbs Italian Herb, 21...	34	bread-crumbs.png
30	Krusteaz No Knead Hawaiian ...	2.45	Delightful Hawaiian Sweet Bread mix, just witho...	21	bread-mix.png
NULL	NULL	NULL	NULL	NULL	NULL

Code explanation:

- Frontend

```
export async function getItems(itemID) {
  let objects = [];
  const query = !itemID ? `http://${url}/item` :
    `http://${url}/item?itemID=${itemID}`;
  await fetch(`${query}`)
    .then((response) => response.json())
    .then((response) => {
      objects = response;
    })
    .catch((err) => {
      console.error(err);
    });
  return objects;
}

export async function getItemInCategory(itemID, categoryID) {
  let objects = [];
  await fetch(`http://${url}/item?itemID=${itemID}&categoryID=${categoryID}`)
    .then((response) => response.json())
    .then((response) => {
      objects = response;
    })
    .catch((err) => {
      console.error(err);
    });
  return objects;
}
```

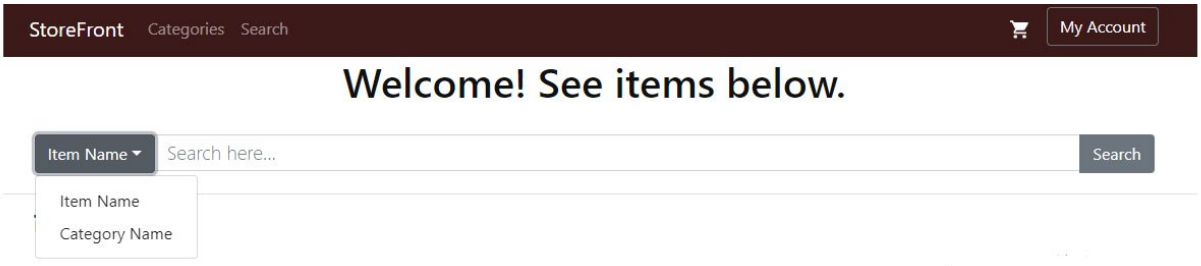
- Backend

```
app.get('/item', (req,res) => {
    const { itemName, itemID, categoryID } = req.query;
    pool.getConnection(function(err, con){
        if(itemName) {
            con.query(`select itemID from Item where itemName like`+
            ` '%${itemName}%'`, (err, results) =>{
                if(err) res.send(err)
                else res.send(results);
            });
        } else if(!itemID && !categoryID) {
            con.query(
                "select c.categoryName, i.itemID, i.itemName, i.price, i.description, i.image, i.quantity" +
                " from belongs inner join Item as i using (itemID) inner join Categories as c using (categoryID) where belongs.categoryID = categoryID and belongs.itemID = itemID;",
                    (err, results) => {
                      //   console.log(error);

                if(err) res.send(err);
                else{
                    res.send(results)
                }
            });
        } else if (!categoryID) {
            con.query(`select * from Item where itemID = ${itemID}`, (err, results) => {
                if (err) res.send(err);
                else {
                    res.send(results);
                }
            });
        } else if (!itemID) {
            con.query(`
                select c.categoryID, c.categoryName, i.itemID, i.itemName, i.price, i.description, i.image, i.quantity
                from belongs inner join Item as i using (itemID) inner join Categories as c using (categoryID) where categoryID = ${categoryID};
            `, (err, results) => {
                if (err) res.send(err);
                else res.send(results);
            });
        } else {
            con.query(`
                select c.categoryID, c.categoryName, i.itemID, i.itemName, i.price, i.description, i.image, i.quantity
                from belongs inner join Item as i using (itemID) inner join Categories as c using (categoryID) where categoryID = ${categoryID} and itemID = ${itemID}
            `, (err, results) => {
                if (err) res.send(err);
                else res.send(results);
            });
        }

        con.release();
    });
});
```

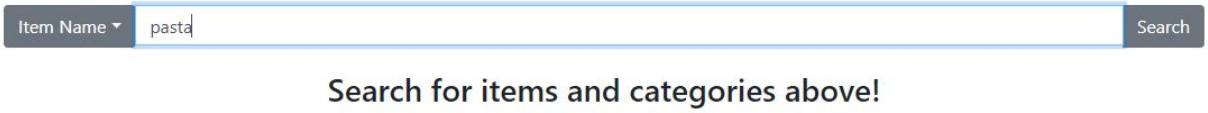
  - The function works based on what is passed into the query. If a name is sent then it will do a comparison. If nothing is sent then it will return a table that has the categories table and items table joined together based on the belongs table. If an itemId is sent then it will return just that item, same with categoryID and categories. If everything is sent then it will return a table containing the category information for that item as well as the item information itself.

Search Items

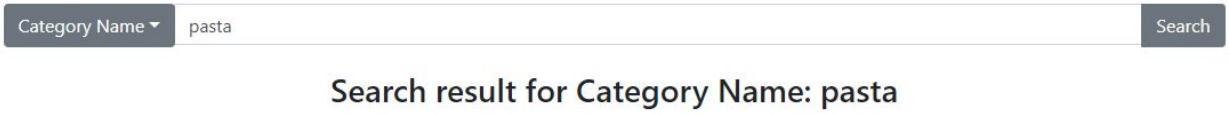
User Interface: One of the basic features that an online webpage usually have is a search function. For our StoreFont, we have the search function that either let the user search by the item name or by the item category.



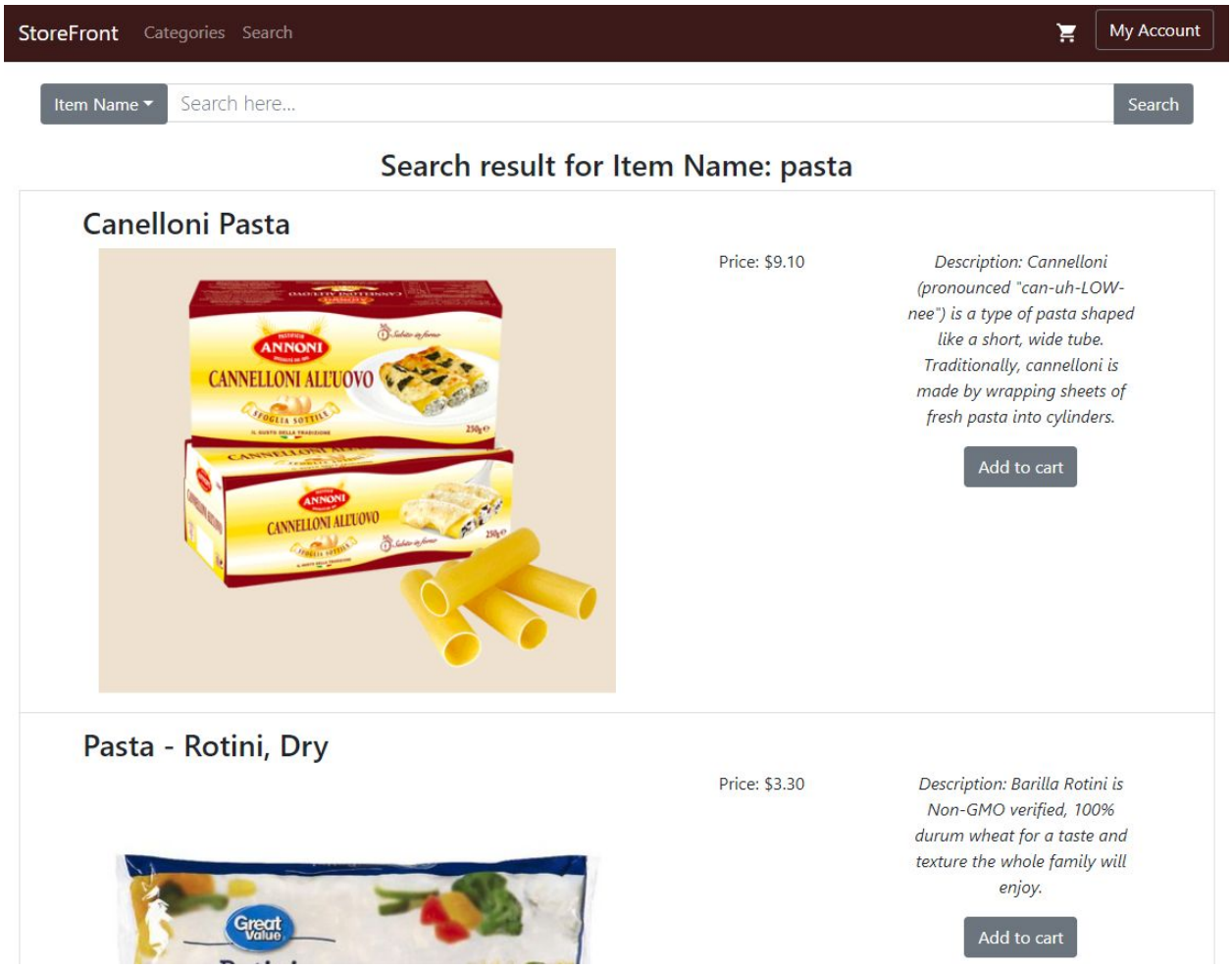
Search by item name:



Or



Result:



Database: to get the result, it will take the user's input and search by either the item's name or category. It using 3 tables, belongs, Item, and Category

Belongs table:

itemID	categoryID
1	11
2	9
3	8
4	8
5	11
6	6
6	10
7	11
8	9
9	12
9	15
10	6
11	7
12	11
13	5
14	3
14	15
15	7
16	1
16	15
17	1
17	15
18	1
18	15
19	2
19	3
20	2
20	3
21	3
21	4
21	15
22	3
22	4
22	15
23	3
23	4
23	15
24	5
24	15
25	3
25	13
25	15
26	1
26	3
26	13
26	15
27	12
28	12
28	15
29	14
30	14
30	15
NULL	NULL

Item table:

itemID	itemName	price	description	quantity	image
1	Duralex Picardie Glass Tumbler	12.9	Standard in French bistros and cafés, classic Pic...	20	glass_item.png
2	Canelloni Pasta	9.1	Cannelloni (pronounced "can-uh-LOW-nee") is a...	50	pasta_item.png
3	Ostrich - Prime Cut	9.1	Ostrich Oyster Steak is one of the most popular...	31	ostrich_item.png
4	Turkey - Ground. Lean	6.3	JENNIE-O® All Natural* Lean Ground Turkey is ...	73	turkey_item.png
5	Doilies Paper	4.5	These white paper doilies are a classic way to s...	50	doilies_item.png
6	Chantrelle Mushroom	23.99	The Chanterelle is a distinctively trumpet-shape...	100	mushroom_item.png
7	Foam Dinner Plate	3.1	White color and great for hot and cold food.	150	plate_item.png
8	Pasta - Rotini, Dry	3.3	Barilla Rotini is Non-GMO verified, 100% durum ...	50	pasta2_item.png
9	Longos - Chicken Curried	7.1	A India inspired soup with chunks of tender chic...	120	curry_item.png
10	Lettuce - Iceberg	2.09	Iceberg is a variety of lettuce with crisp leaves ...	50	letture_item.png
11	Wine - Cotes Du Rhone	15.1	Medium garnet color, with bright red reflections...	160	wine_item.png
12	Garbage Bag - Clear	4.75	The only tall kitchen recycling bags with odor co...	50	trashbag_item.png
13	Pumpkin	5.1	A pumpkin is a cultivar of a squash plant, most c...	50	pumpkin_item.png
14	Piping Jelly - All Colours	25.54	Highly concentrated vibrant colors mix easily an...	40	icying_item.png
15	Wine - Zonnebloem Pinotage	49.99	A complex wine with red fruit flavors. Big, elega...	50	wine2_item.png
16	Cheez-It Baked Snack Crackers	4.67	Baked Snack Crackers, Original Cheddar, Family...	81	cheezIt.png
17	22CT GRAB & SNACK	10	Bring your favorite Wise snacks with you where...	18	variety_packs.png
18	Pringles Potato Crisps Original...	1.54	Satisfy your snack craving with the salty, stack...	62	potato-chips.png
19	Breyers Frozen Dairy Dessert	4.49	Oreo cookie pieces in vanilla flavored frozen dai...	33	breyers-ice-cream.png
20	Dairy Free Frozen Dessert Ca...	4.69	Rich, creamy & totally indulgent! Certified glute...	21	cashew-milk-ice-cream.png
21	Mountain Dew Soda	26.57	Pack of 24, 12 ounces.cans. 170 calories per ca...	35	mountain-dew-soda.png
22	Zevia Zero Calorie Soda, Rain...	22.14	MORE THAN A BEVERAGE: Zevia is a movement...	42	zevia-soda.png
23	Fanta Orange Soda Fruit Flav...	6.66	Pack of twelve, 12 FL OZ per can of naturally fl...	32	fanta-soda.png
24	DOLE Sliced Yellow Cling Peac...	2.88	HEALTHY SNACKS AND JUICES: From packaged...	23	dole-peaches.png
25	Jell-O No Bake Cherry Chees...	3.14	One 17.8 oz. package of Jell-O No Bake Cherry ...	34	chesse-cake.png
26	Quest Nutrition Protein Bar, B...	0.99	Craveable Birthday Cake flavored coated protei...	100	cake-bar.png
27	Nescafe Taster's Choice Insta...	2.99	Made for an 8 ounce Cup, each of the 16 conve...	99	instant-coffee.png
28	Maruchan Ramen Noodles Chi...	3.5	Maruchan Souper 6 Pack Ramen Noodle Soup C...	88	instant-noodle.png
29	McCormick Panko Bread Crum...	7.87	McCormick Panko Bread Crumbs Italian Herb, 21...	34	bread-crumbs.png
30	Krusteaz No Knead Hawaiian ...	2.45	Delightful Hawaiian Sweet Bread mix, just witho...	21	bread-mix.png
NULL	NULL	NULL	NULL	NULL	NULL

Category:

categoryID	categoryName
1	Snacks
2	Ice Cream
3	Sweet
4	Soda
5	Fruit
6	Veggies
7	Wine
8	Meat
9	Pasta
10	Mushroom
11	Kitchen Utensils
12	Instant Food
13	Cake
14	Bread
15	Processed Food
NULL	NULL

Code explanation:

- Frontend

```
export async function searchByItemName(itemName) {
  let objects = [];
  let itemIDs = [];
  await fetch(`http://${url}/item?itemName=${itemName}`)
    .then((response) => response.json())
    .then(async (response) => {
      response.map(x => itemIDs.push(x.itemID));
      await Promise.all(itemIDs.map(async (id) => {
        let currentItem = await getItems(id);
        objects = objects.concat(currentItem);
      }));
    })
    .catch((err) => {
      console.error(err);
    });
  return objects;
}
```

- Backend
  - Works the same as the “Browse” function

```
app.get('/item', (req,res) => {
    const { itemName, itemID, categoryID } = req.query;
    pool.getConnection(function(err, con){
        if(itemName) {
            con.query(`select itemID from Item where itemName like`+
            ` '%${itemName}%'`, (err, results) =>{
                if(err) res.send(err)
                else res.send(results);
            });
        } else if(!itemID && !categoryID) {
            con.query(
                "select c.categoryName, i.itemID, i.itemName, i.price, i.description, i.image, i.quantity" +
                " from belongs inner join Item as i using (itemID) inner join Categories as c using (categoryID) where belongs.categoryID = categoryID and belongs.itemID = itemID;",
                    (err, results) => {
                      //   console.log(error);

                if(err) res.send(err);
                else{
                    res.send(results)
                }
            });
        } else if (!categoryID) {
            con.query(`select * from Item where itemID = ${itemID}`, (err, results) => {
                if (err) res.send(err);
                else {
                    res.send(results);
                }
            });
        } else if (!itemID) {
            con.query(`
                select c.categoryID, c.categoryName, i.itemID, i.itemName, i.price, i.description, i.image, i.quantity
                from belongs inner join Item as i using (itemID) inner join Categories as c using (categoryID) where categoryID = ${categoryID};
```

```
                `, (err, results) => {
                  if (err) res.send(err);
                  else res.send(results);
                });
            } else {
                con.query(`
                    select c.categoryID, c.categoryName, i.itemID, i.itemName, i.price, i.description, i.image, i.quantity
                    from belongs inner join Item as i using (itemID) inner join Categories as c using (categoryID) where categoryID = ${categoryID} and itemID = ${itemID}
                `, (err, results) => {
                    if (err) res.send(err);
                    else res.send(results);
                });
            }

            con.release();
        });
    });
```

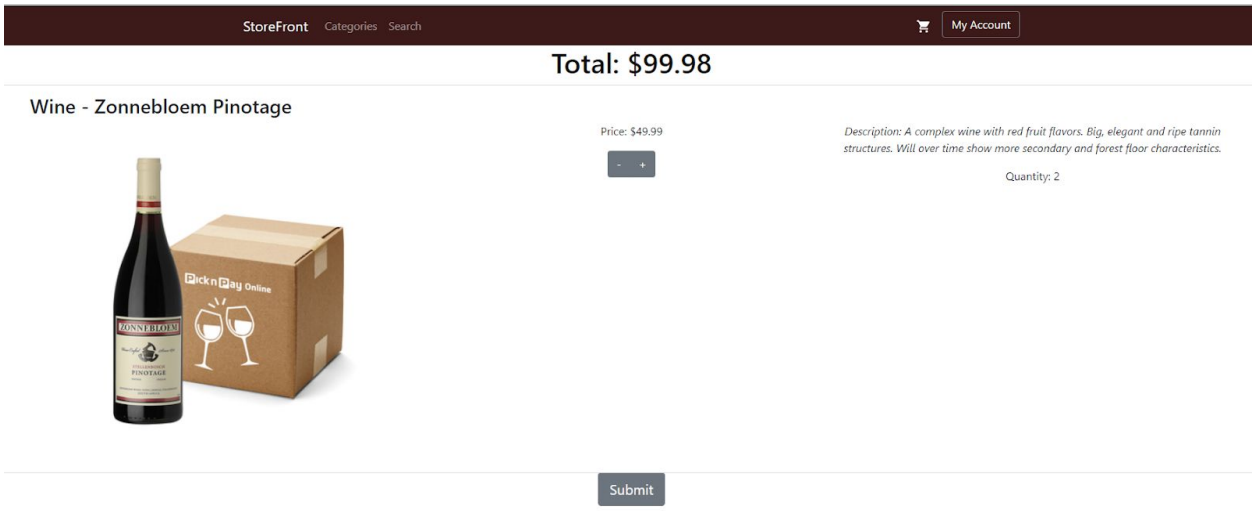
View Cart

User Interface: the user can also view the items that are currently in their cart

To get to the cart page, click on the shopping cart icon

When cart is empty:

When cart contains some items:



Database: the database will make use of the Cart table and the holds relationship table

When cart is empty (note that our example account is 21 and in the images below there are no accountID that equal to 21):

```
5 •  SELECT * FROM Storefront.Cart;
```

esult Grid | Filter Rows:

accountID	itemID	quantity
1	1	1
1	2	1
1	4	3
2	1	1
5	1	2
5	6	1
6	7	3
6	9	1
6	10	1
7	8	1
7	9	2
8	1	1
8	9	2
10	2	1
10	3	2
10	5	1
10	6	1
NULL	NULL	NULL

When cart contains some items(note that our example account is 21):

```
5 •  SELECT * FROM Storefront.Cart;
```

esult Grid | Filter Rows:

accountID	itemID	quantity
1	1	1
1	2	1
1	4	3
2	1	1
5	1	2
5	6	1
6	7	3
6	9	1
6	10	1
7	8	1
7	9	2
8	1	1
8	9	2
10	2	1
10	3	2
10	5	1
10	6	1
21	15	2
NULL	NULL	NULL

Will we check, the item id of 15, it is indeed wine

```
5 •   SELECT * FROM Storefront.Item;
```

esult Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

itemID	itemName	price	description	quantity	image
8	Pasta - Rotini, Dry	3.3	Barilla Rotini is Non-GMO verified, 100% durum ...	50	pasta2_item.png
9	Longos - Chicken Curried	7.1	A India inspired soup with chunks of tender chic...	120	curry_item.png
10	Lettuce - Iceberg	2.09	Iceberg is a variety of lettuce with crisp leaves ...	50	letture_item.png
11	Wine - Cotes Du Rhone	15.1	Medium garnet color, with bright red reflections...	160	wine_item.png
12	Garbage Bag - Clear	4.75	The only tall kitchen recycling bags with odor co...	50	trashbag_item.png
13	Pumpkin	5.1	A pumpkin is a cultivar of a squash plant, most c...	50	pumpkin_item.png
14	Piping Jelly - All Colours	25.54	Highly concentrated vibrant colors mix easily an...	40	icying_item.png
15	Wine - Zonnebloem Pinotage	49.99	A complex wine with red fruit flavors. Big, elega...	50	wine2_item.png
16	Cheez-It Baked Snack Crackers	4.67	Baked Snack Crackers, Original Cheddar, Family...	81	cheezIt.png

Code explanation:

- Frontend

```
export async function getCartItems(accountID) {
  let objects = [];
  await fetch(`http://${url}/cart?accountID=${accountID}`)
    .then((response) => response.json())
    .then((response) => {
      objects = response;
    })
    .catch((err) => {
      console.error(err);
    });
  return objects;
}
```

- Backend

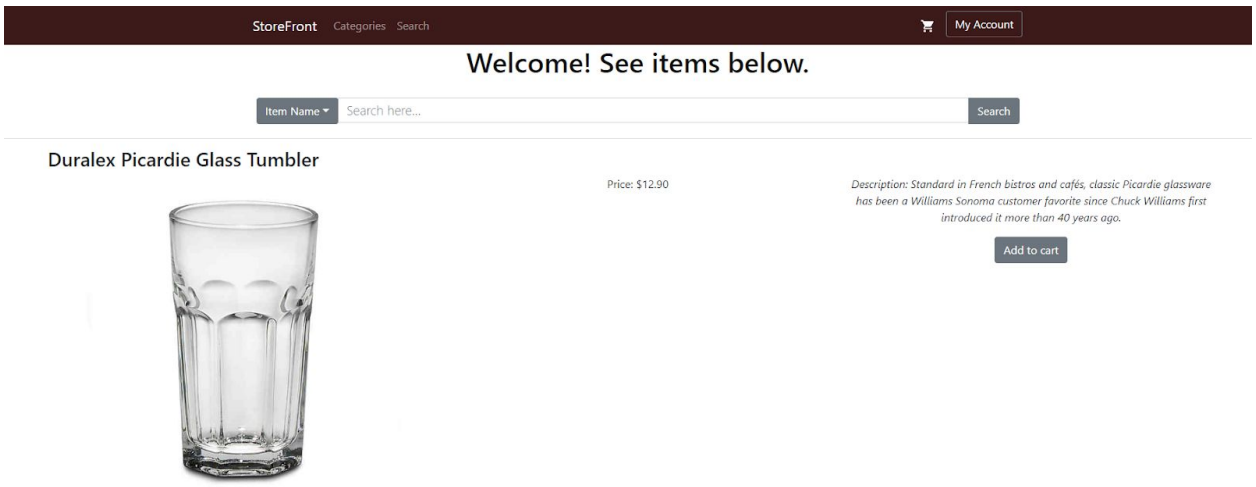
```
app.get('/cart', (req, res) => {
    const { accountID } = req.query;
    pool.getConnection(function (err, con) {
        con.query(`select * from Cart where accountID=${accountID}`, (err, results) => {
            if (err) res.send(err);
            else {
                res.send(results);
            }
        });
        con.release();
    });
});
```

  - Simply send all tuples that contains the accountID that is sent to the function.

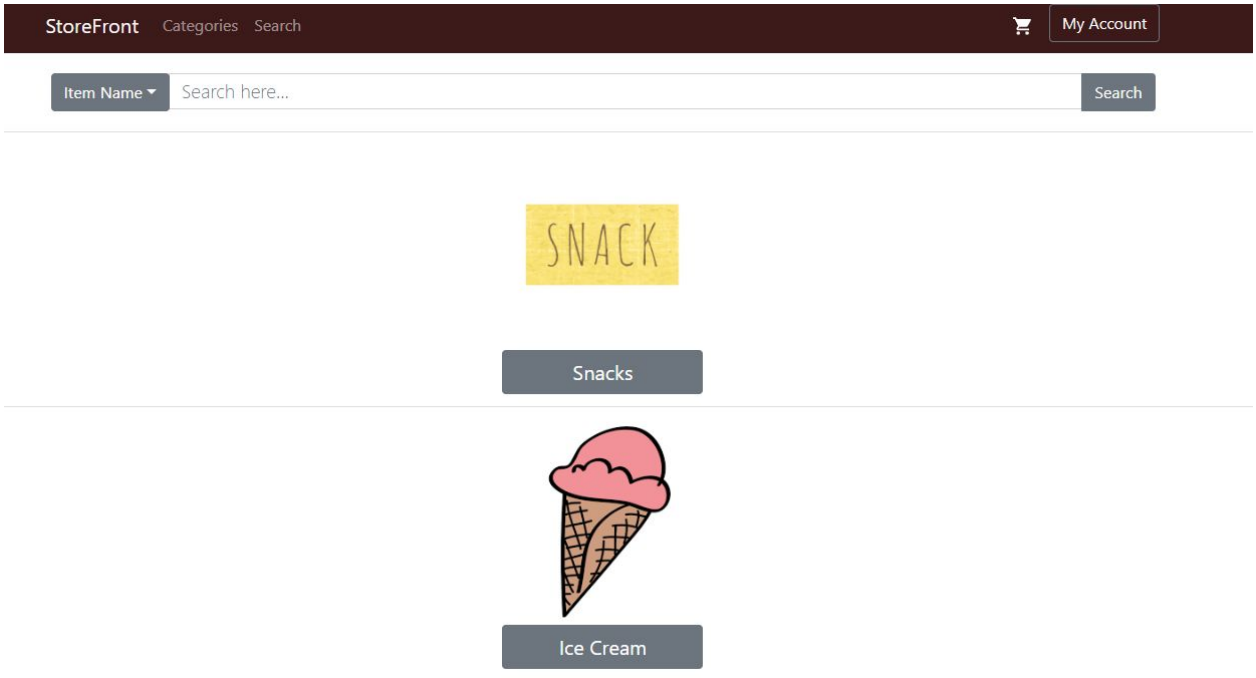
Adding items to the cart

User Interface: there are a few ways for the user to add items to the cart.

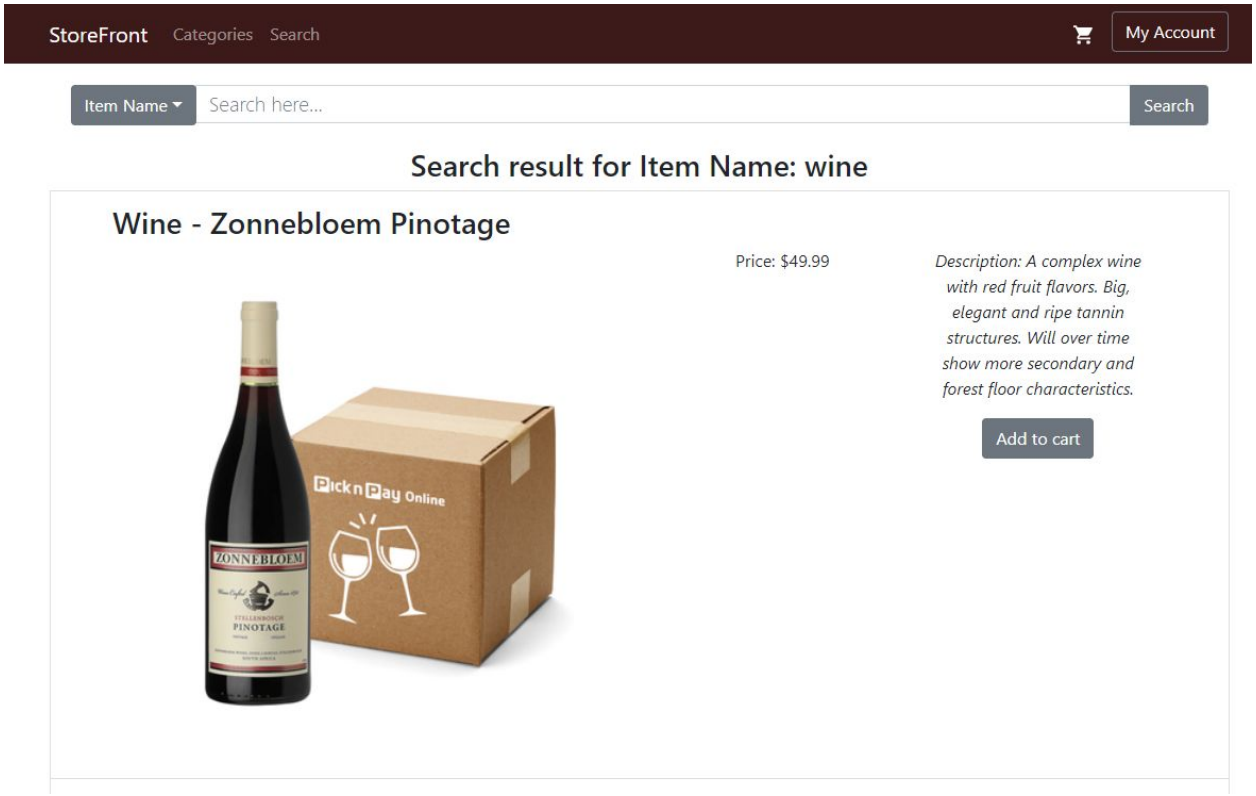
The 1st one is by selecting the item from the browsing page



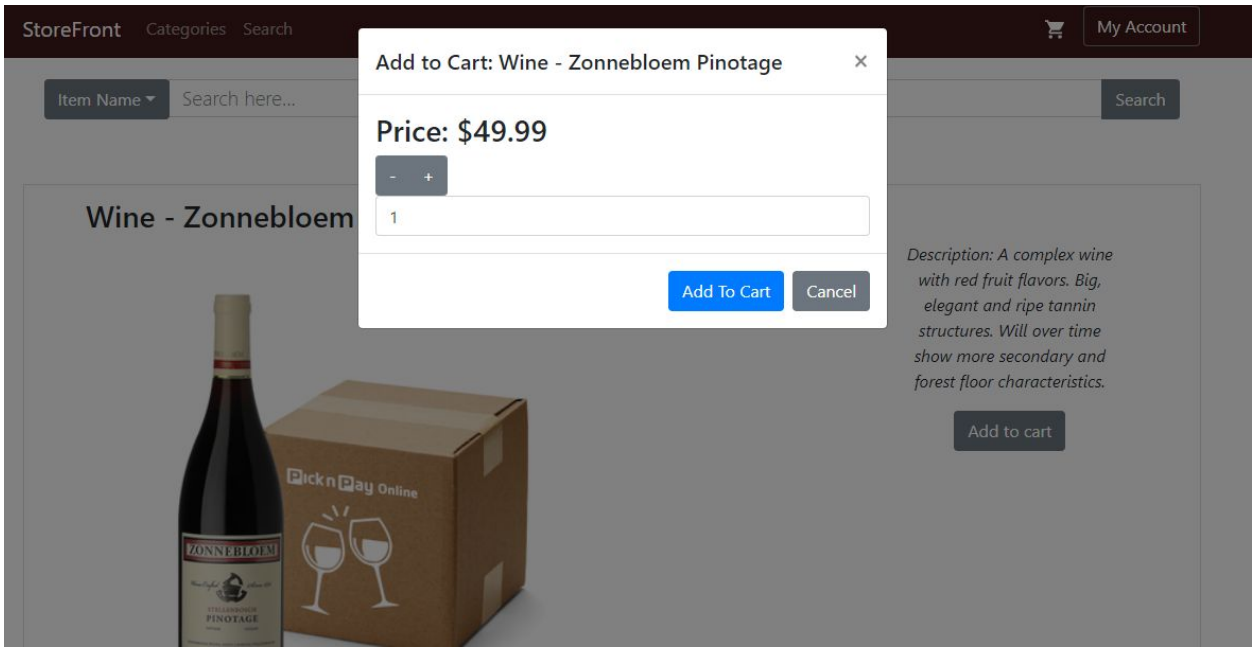
The 2nd one is by picking items in a specific category



The 3rd way is to search directly for a specific item:

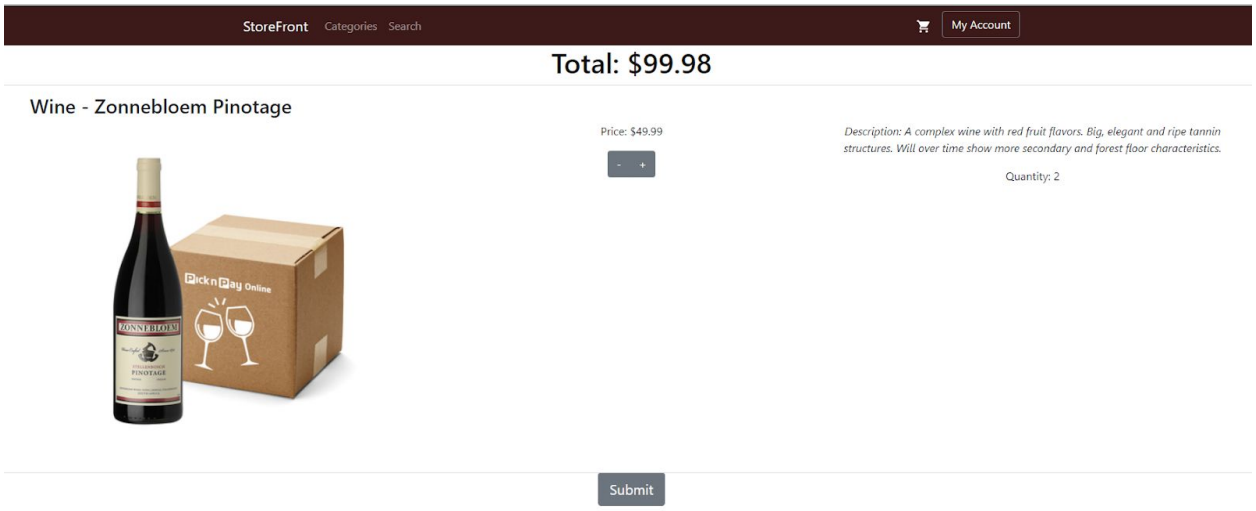


After that, we can add the item to the cart:



The cart before add item:

The cart after add item:



Database: after the user has picked the item and the amount of quantity that they want, they can click on the “Add to Cart” button and it will save the data to the database.

Before the “Add to Cart” button is clicked:
When cart is empty (note that our example account is 21 and in the images below there are no accountID that equal to 21):

```
5 •  SELECT * FROM Storefront.Cart;
```

esult Grid | Filter Rows:

accountID	itemID	quantity
1	1	1
1	2	1
1	4	3
2	1	1
5	1	2
5	6	1
6	7	3
6	9	1
6	10	1
7	8	1
7	9	2
8	1	1
8	9	2
10	2	1
10	3	2
10	5	1
10	6	1
NULL	NULL	NULL

When cart contains some items(note that our example account is 21):

```
5 •  SELECT * FROM Storefront.Cart;
```

esult Grid | Filter Rows:

accountID	itemID	quantity
1	1	1
1	2	1
1	4	3
2	1	1
5	1	2
5	6	1
6	7	3
6	9	1
6	10	1
7	8	1
7	9	2
8	1	1
8	9	2
10	2	1
10	3	2
10	5	1
10	6	1
21	15	2
NULL	NULL	NULL

Code explanation:

- Frontend

```
export async function addToCart(accountID, itemID, quantity) {
  let objects = [];
  await fetch(`http://${url}/cart/add?accountID=${accountID}` +
    `&itemID=${itemID}&quantity=${quantity}`)
    .then((response) => response.json())
    .then((response) => {
      objects = response;
    })
    .catch((err) => {
      console.error(err);
    });
  return objects;
}
```

- Backend

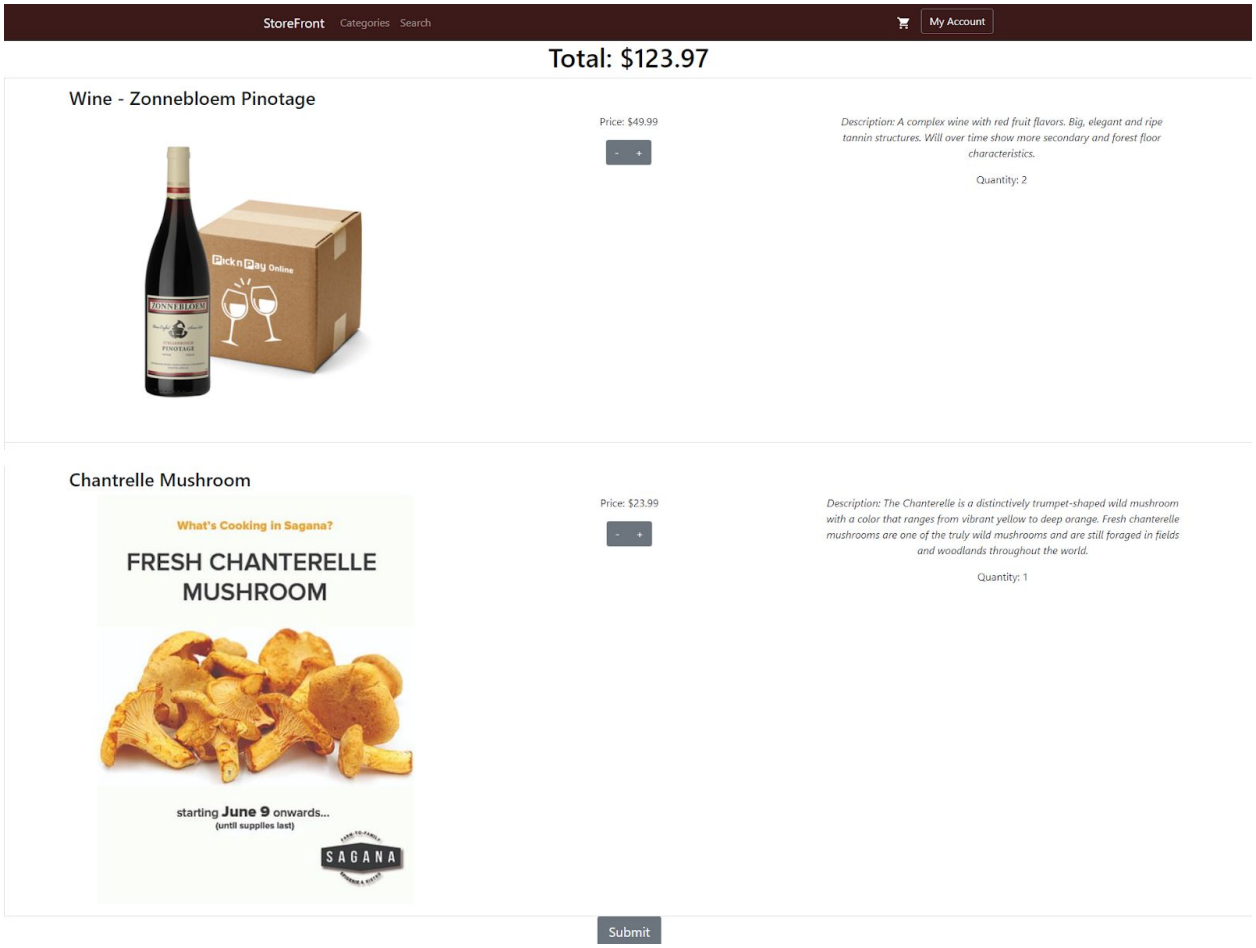
```
app.get('/cart/add', (req, res) => {
    const { accountID, itemID, quantity } = req.query;
    pool.getConnection(function (err, con) {
        con.query(`select * from Cart where accountID=${accountID}
        and itemID=${itemID}`, (err, results) => {
            if (err) res.send(err);
            else {
                if (results.length == 1) {
                    con.query(`update Cart set quantity =
                    quantity + ${quantity} where accountID=${accountID}
                    and itemID=${itemID}`, (err, results) => {
                        if (err) res.send(err);
                        else res.send(results);
                    });
                } else {
                    con.query(`
                                insert into
                                    Cart(accountID, itemID, quantity)
                                values(${accountID},${itemID},${quantity})`, (err, results) => {
                        if (err) res.send(err);
                        else res.send(results);
                    });
                }
            }
        });
        con.release();
    });
});
```

    - Updates cart information for each item. If the item is found then it will update the quantity, otherwise it will insert a new item into the database into the cart table.

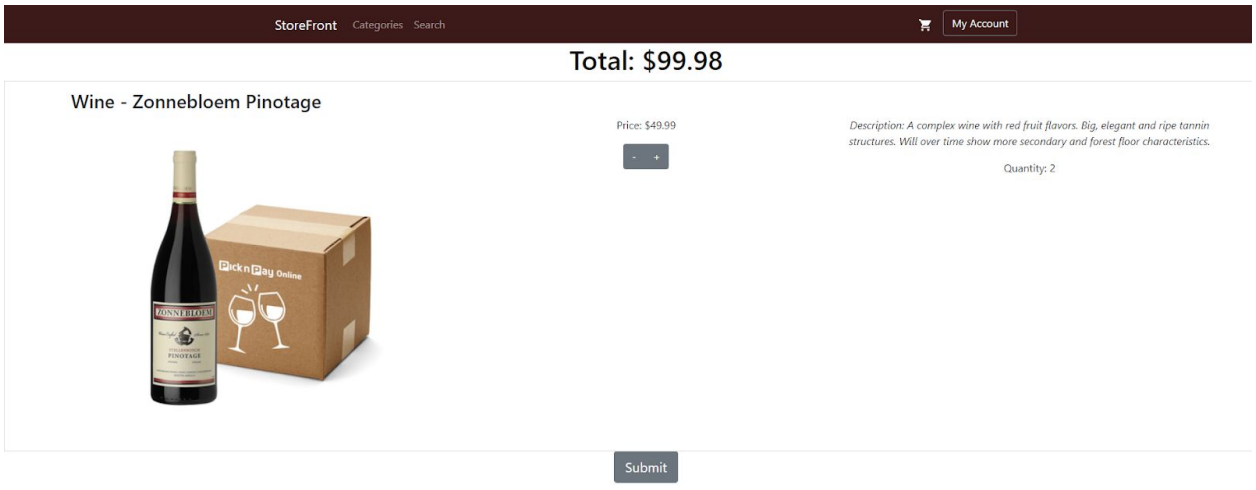
Deleting items in the cart

User Interface: the user can also remove a particular item that they don't want from the cart

Start off by looking at the CartPage



After removing an item by decreasing the amount of item to zero, for example, Chanterelle Mushroom, now the CartPage will look like this:



Database: the query will make use of the Cart table

Before an item is removed.

```
5 •  SELECT * FROM Storefront.Cart;
```

esult Grid | Filter Rows:

accountID	itemID	quantity
1	1	1
1	2	1
1	4	3
2	1	1
5	1	2
5	6	1
6	7	3
6	9	1
6	10	1
7	8	1
7	9	2
8	1	1
8	9	2
10	2	1
10	3	2
10	5	1
10	6	1
21	6	1
21	15	2
NULL	NULL	NULL

After the item is removed by decreasing the amount of the particular item to zero (in this example, we will remove the itemID 6 that belongs to accountID 21):

```
5 •  SELECT * FROM Storefront.Cart;
```

esult Grid | Filter Rows:

accountID	itemID	quantity
1	1	1
1	2	1
1	4	3
2	1	1
5	1	2
5	6	1
6	7	3
6	9	1
6	10	1
7	8	1
7	9	2
8	1	1
8	9	2
10	2	1
10	3	2
10	5	1
10	6	1
21	15	2
NULL	NULL	NULL

Code explanation:

- Frontend

```
export async function removeFromCart(accountID, itemID, quantity) {
  let objects = [];
  await fetch(`http://${url}/cart/remove?accountID=${accountID}` +
    `&itemID=${itemID}&quantity=${quantity}`)
    .then((response) => response.json())
    .then((response) => {
      objects = response;
    })
    .catch((err) => {
      console.error(err);
    });
  return objects;
}
```

- Backend

```
app.get('/cart/remove', (req, res) => {
    const { accountID, itemID, quantity } = req.query;
    pool.getConnection(function (err, con) {
        con.query(`select * from Cart where accountID=${accountID}
        and itemID=${itemID}`, (err, results) => {
            if (err) res.send(err);
            else {
                let currentQuantity = parseInt(results[0].quantity);
                if (currentQuantity === 1) {
                    con.query(`delete from Cart where accountID=${accountID}
                    and itemID=${itemID}`, (err, results) => {
                        if (err) res.send(err);
                        else res.send(results);
                    });
                } else {
                    con.query(`update Cart set quantity =
                    ${currentQuantity - quantity} where accountID=${accountID}
                    and itemID=${itemID}`, (err, results) => {
                        if (err) res.send(err);
                        else res.send(results);
                    });
                }
            }
        });
        con.release();
    });
});
```

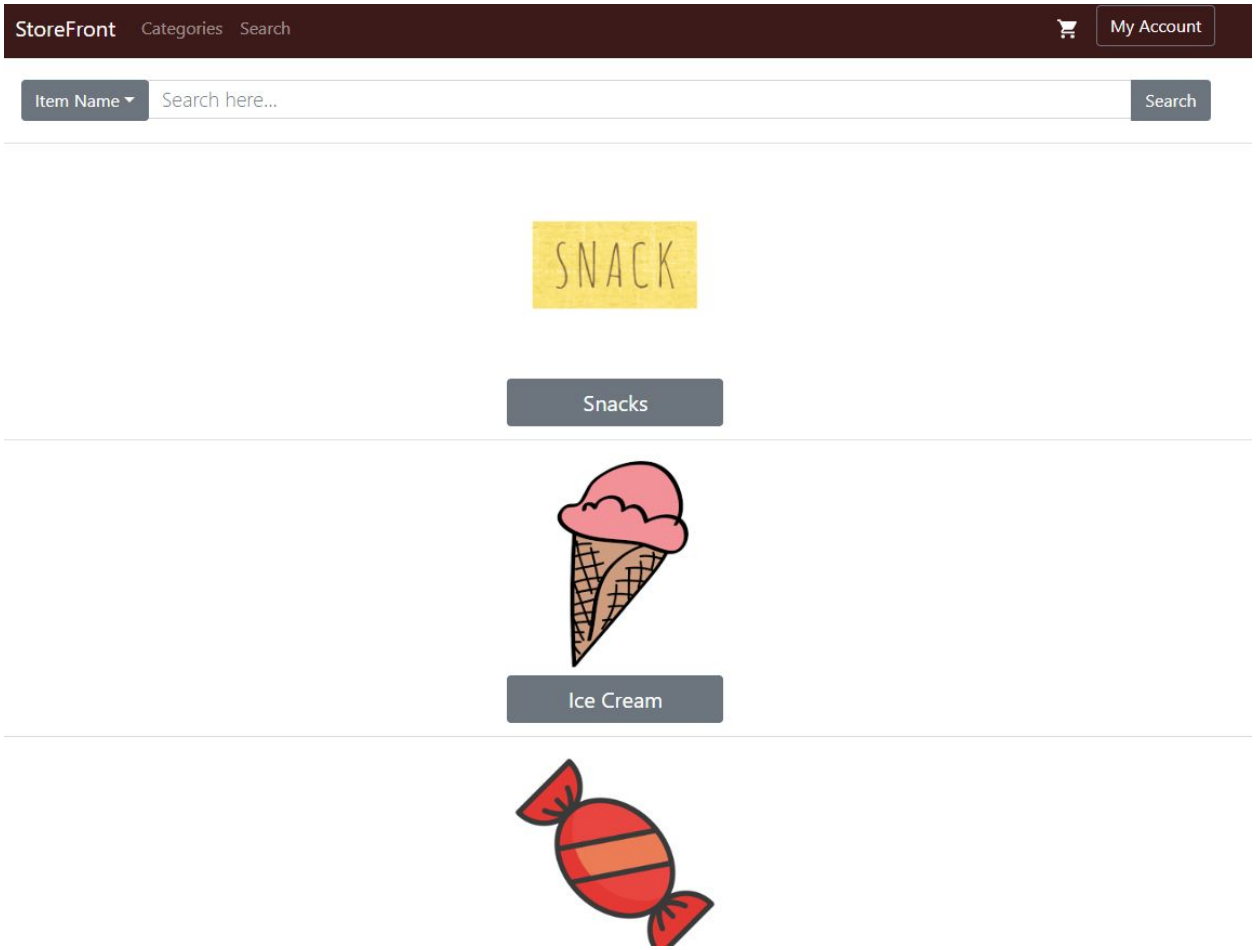
    - Selects the tuples in the cart where the account ID is found and decrements or removes a specified amount from the quantities.

View Category

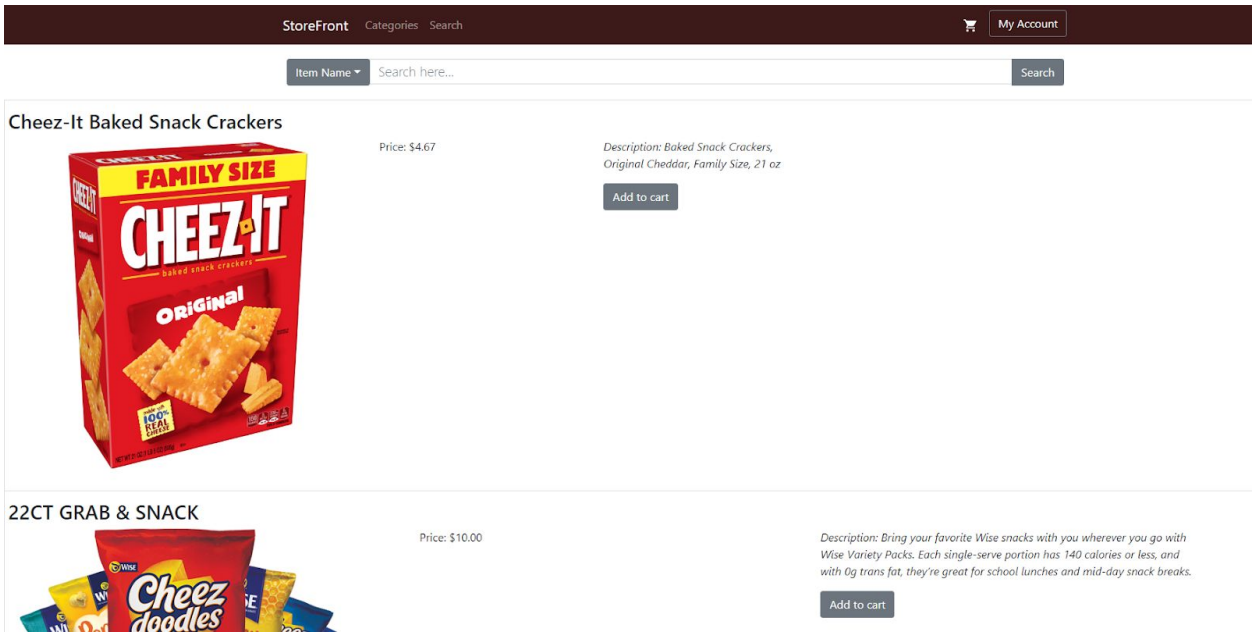
User Interface: the users also have an option to browse the item by category

First, they need to go to the categoryPage which is located on the left hand side on the navigation bar:

By clicking on the “Categories” tab, the user will be able to see this:



After that, they can view all the items below to a specific category:



Database: in order to display the category page, the query will involving the Category table and possibly the it also include the Item table and its relationship table named belongs

```
5 •  SELECT * FROM Storefront.Categories;
```

esult Grid | Filter Rows: | Edit:

categoryID	categoryName
1	Snacks
2	Ice Cream
3	Sweet
4	Soda
5	Fruit
6	Veggies
7	Wine
8	Meat
9	Pasta
10	Mushroom
11	Kitchen Utensils
12	Instant Food
13	Cake
14	Bread
15	Processed Food
NULL	NULL

```
5 •  SELECT * FROM Storefront.belongs;
```

sult Grid | Filter Rows: | Edit:

itemID	categoryID
1	11
2	9
3	8
4	8
5	11
6	6
6	10
7	11
8	9
9	12
9	15
10	6
11	7
12	11
13	5
14	3
14	15
15	7
16	1
16	15
17	1
17	15
18	1
18	15
19	2
19	3
20	2
20	3
21	3
21	4
21	15
22	3
22	4
22	15
23	3
23	4
23	15
24	5
24	15
25	3
25	13
25	15
26	1
26	3
26	13
26	15
28	12
28	15
29	14
30	14
30	15
NULL	NULL

```
5 •  SELECT * FROM Storefront.Item;
```

esult Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

itemID	itemName	price	description	quantity	image
1	Duralex Picardie Glass Tumbler	12.9	Standard in French bistros and cafés, classic Pic...	20	glass_item.png
2	Canelloni Pasta	9.1	Cannelloni (pronounced "can-uh-LOW-nee") is a...	50	pasta_item.png
3	Ostrich - Prime Cut	9.1	Ostrich Oyster Steak is one of the most popular...	31	ostrich_item.png
4	Turkey - Ground. Lean	6.3	JENNIE-O® All Natural* Lean Ground Turkey is ...	73	turkey_item.png
5	Doilies Paper	4.5	These white paper doilies are a classic way to s...	50	doilies_item.png
6	Chantrelle Mushroom	23.99	The Chanterelle is a distinctively trumpet-shape...	100	mushroom_item.png
7	Foam Dinner Plate	3.1	White color and great for hot and cold food.	150	plate_item.png
8	Pasta - Rotini, Dry	3.3	Barilla Rotini is Non-GMO verified, 100% durum ...	50	pasta2_item.png
9	Longos - Chicken Curried	7.1	A India inspired soup with chunks of tender chic...	120	curry_item.png
10	Lettuce - Iceberg	2.09	Iceberg is a variety of lettuce with crisp leaves ...	50	letture_item.png
11	Wine - Cotes Du Rhone	15.1	Medium garnet color, with bright red reflections...	160	wine_item.png
12	Garbage Bag - Clear	4.75	The only tall kitchen recycling bags with odor co...	50	trashbag_item.png
13	Pumpkin	5.1	A pumpkin is a cultivar of a squash plant, most c...	50	pumpkin_item.png
14	Piping Jelly - All Colours	25.54	Highly concentrated vibrant colors mix easily an...	40	icying_item.png
15	Wine - Zonnebloem Pinotage	49.99	A complex wine with red fruit flavors. Big, elega...	50	wine2_item.png
16	Cheez-It Baked Snack Crackers	4.67	Baked Snack Crackers, Original Cheddar, Family...	81	cheezIt.png
17	22CT GRAB & SNACK	10	Bring your favorite Wise snacks with you where...	18	variety_packs.png
18	Pringles Potato Crisps Original...	1.54	Satisfy your snack craving with the salty, stack...	62	potato-chips.png
19	Breyers Frozen Dairy Dessert	4.49	Oreo cookie pieces in vanilla flavored frozen dai...	33	breyers-ice-cream.png
20	Dairy Free Frozen Dessert Ca...	4.69	Rich, creamy & totally indulgent! Certified glute...	21	cashew-milk-ice-cream.png
21	Mountain Dew Soda	26.57	Pack of 24, 12 ounces.cans. 170 calories per ca...	35	mountain-dew-soda.png
22	Zevia Zero Calorie Soda, Rain...	22.14	MORE THAN A BEVERAGE: Zevia is a movement...	42	zevia-soda.png
23	Fanta Orange Soda Fruit Flav...	6.66	Pack of twelve, 12 FL OZ per can of naturally fl...	32	fanta-soda.png
24	DOLE Sliced Yellow Cling Peac...	2.88	HEALTHY SNACKS AND JUICES: From packaged...	23	dole-peaches.png
25	Jell-O No Bake Cherry Chees...	3.14	One 17.8 oz. package of Jell-O No Bake Cherry ...	34	chesse-cake.png
26	Quest Nutrition Protein Bar, B...	0.99	Craveable Birthday Cake flavored coated protei...	100	cake-bar.png
27	Nescafe Taster's Choice Insta...	2.99	Made for an 8 ounce Cup, each of the 16 conve...	99	instant-coffee.png
28	Maruchan Ramen Noodles Chi...	3.5	Maruchan Souper 6 Pack Ramen Noodle Soup C...	88	instant-noodle.png
29	McCormick Panko Bread Crum...	7.87	McCormick Panko Bread Crumbs Italian Herb, 21...	34	bread-crumbs.png
30	Krusteaz No Knead Hawaiian ...	2.45	Delightful Hawaiian Sweet Bread mix, just witho...	21	bread-mix.png
NULL	NULL	NULL	NULL	NULL	NULL

Code explanation:

- Frontend

```
export async function getCategory(categoryID) {
  let objects = [];
  const query = !categoryID ? `http://${url}/item/categories` :
    `http://${url}/item?categoryID=${categoryID}`;

  await fetch(`${query}`)
    .then((response) => response.json())
    .then((response) => {
      objects = response;
    })
    .catch((err) => {
      console.error(err);
    });
  return objects;
}
```

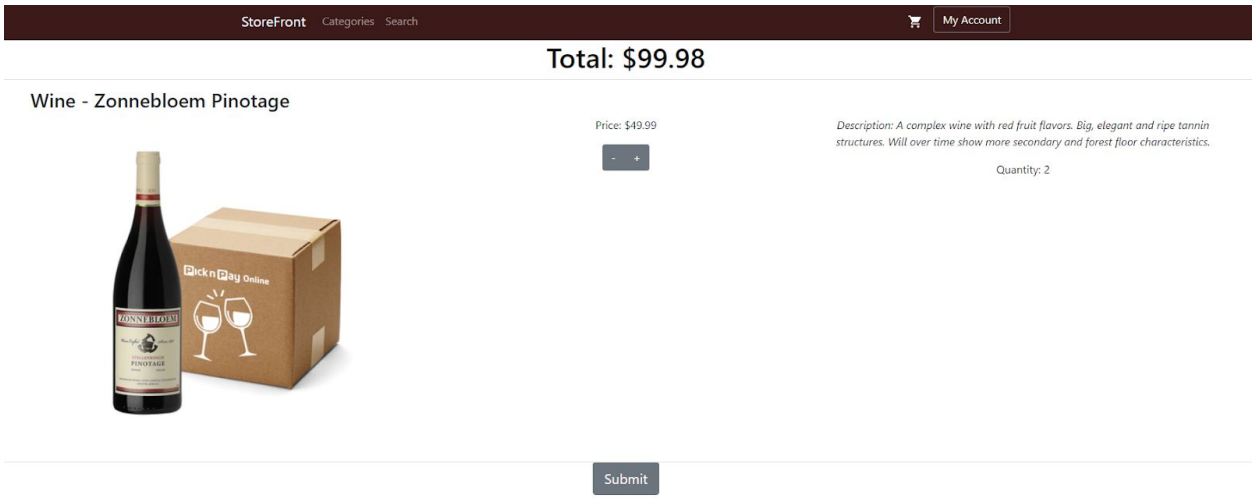
- Backend
    - For all categories we have a different endpoint than if a search is done. If a search is done then we will only return similar category names

```
app.get('/item/categories', (req, res) => {
    // const { categoryID } = req.query;
    pool.getConnection(function (err, con) {
        con.query(`select * from Categories`, (err, results) => {
            if (err) res.send(err)
            else res.send(results);
        });
        con.release();
    });
});

app.get('/category', (req,res) => {
    const { categoryName } = req.query;
    pool.getConnection(function(err, con) {
        con.query(`select categoryID from Categories where categoryName like`+
        ` '%${categoryName}%'`, (err, results) =>{
            if(err) res.send(err)
            else res.send(results);
        });
        con.release();
    });
});
```

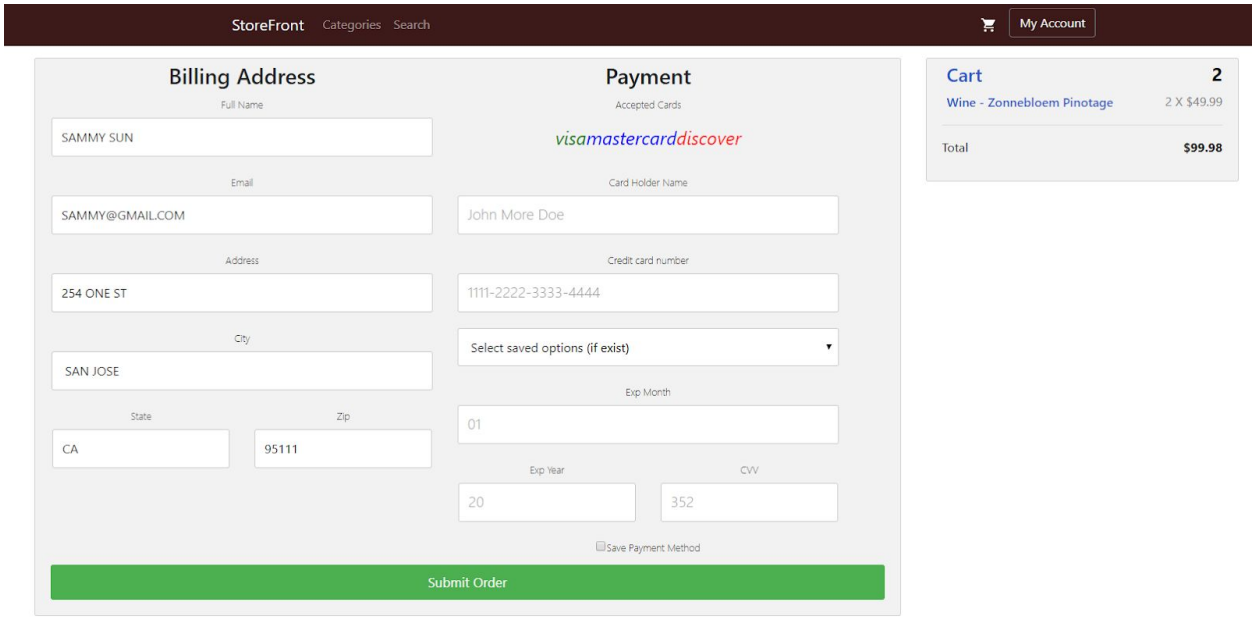
Adding new payment methods

User Interface: the webpage can also store the user's credit card payment methods for the user future reference

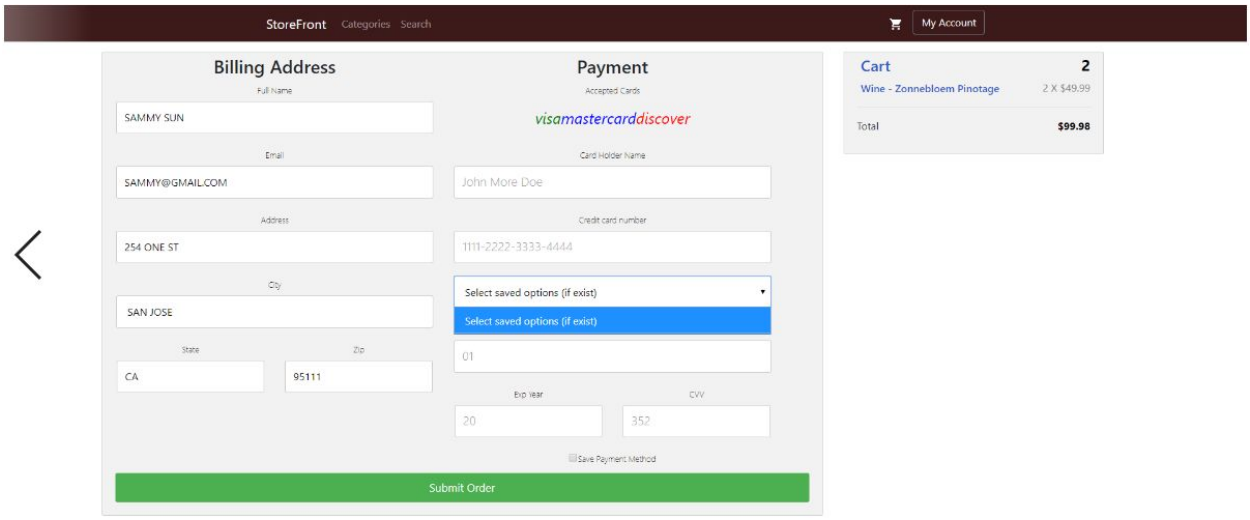
In order to get to the add new payment methods payment, the user needs to check out by clicking on the submit button located at the bottom of the Cart page.



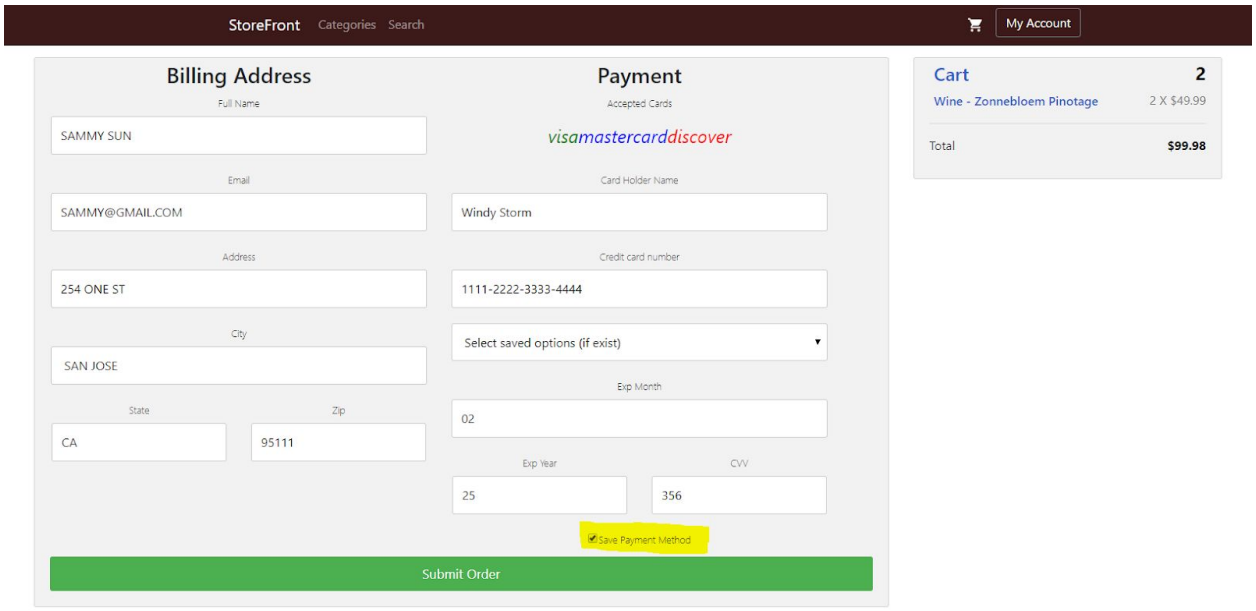
Then it will lead the user to the payment page



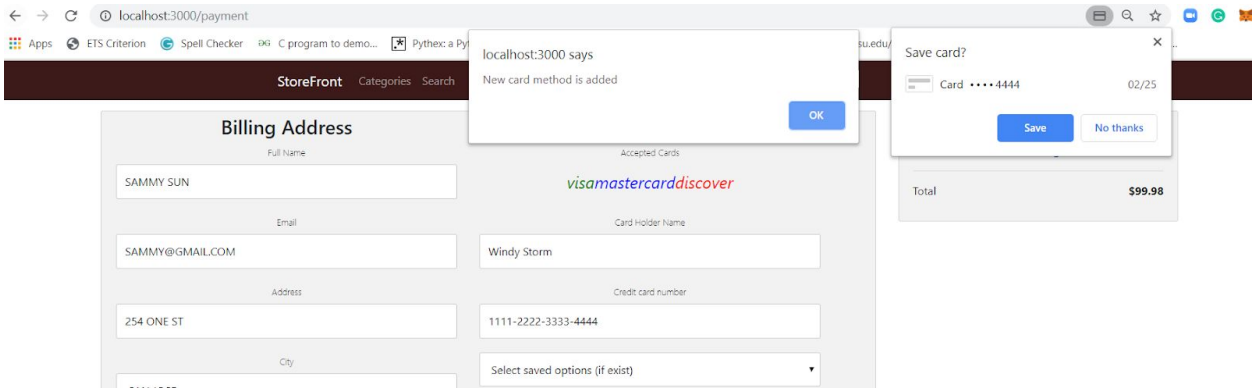
On this payment page, it will automatically fill in the user's billing address information. They can either select on the already existing payment method or create a new one.



After fill in and passing all the check of all the necessary information for the new payment method, in order to let the pay know that the user wants to save the credit card information as a new card payment. They need to click on the box where it says “Save Payment Method”.



When click on the “Submit Order”, it is when the payment method is saved



Database: When save the new payment method, the query involved the CardInfo table and the relationship table named holds.

Before new payment is added:

```
5 •  SELECT * FROM Storefront.CardInfo;
```

sult Grid | Filter Rows: | Edit: | Export/Import:

cardID	CardHolder	CVV	Zip	CardNumber	ExpMonth	ExpYear
1	Samuel Roger	845	974512	4547-2215-5589-7865	02	24
2	Thomas Austin	784	98541	5214-2015-5632-4512	05	22
3	Keith Jam	124	95115	3564-1245-7841-8485	04	20
4	Gena Hopewell	168	93213	3539-1283-2029-5838	12	22
5	Bob Dang	985	98511	7454-1245-5457-6932	08	25
6	Cinderella McRitchie	212	93202	5002-3566-6226-9946	12	22
7	Hildagard Symington	651	95075	5610-4590-1040-5420	12	21
8	Eddy Ung	451	96235	7452-1254-4755-6542	01	27
9	Chiquia Dennington	643	91256	5183-0293-8061-3577	09	24
10	Kalindi Schubart	559	95653	3565-2575-2781-0847	07	23
11	Ronny Sa	541	97512	4564-2535-5667-2633	05	24
12	Odell McNaught	692	94068	3536-4509-0939-8434	01	21
13	Michele Sawl	623	92195	3567-3354-4948-4301	01	25
14	Aili Rean	273	91062	4211-9517-2088-6036	11	23
15	Heddi Esposita	518	91799	2018-8097-9437-1003	11	26
16	Loreen McIndoe	685	94621	3580-1560-6271-4789	12	24
17	Clea Allright	778	94068	2723-0115-9528-8936	06	22
18	Isaac Marlowe	685	92035	3038-1288-3039-8734	07	25
19	Alica Legh	303	94910	5038-9081-6431-0355	02	24
20	Tom Cruz	456	95132	4575-8956-4521-1254	11	27
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
5 •  SELECT * FROM Storefront.holds;
```

esult Grid | Filter Rows:

accountID	cardID
2	4
3	5
5	6
6	7
7	8
7	9
7	10
11	11
12	12
13	13
14	15
14	16
14	17
16	14
18	18
19	19
20	20
NULL	NULL

After the new payment method is added, the new card information is already there which in this case is cardID 21:

```
5 •  SELECT * FROM Storefront.CardInfo;
```

esult Grid | Filter Rows: | Edit: | Export/Import:

cardID	CardHolder	CVV	Zip	CardNumber	ExpMonth	ExpYear
1	Samuel Roger	845	974512	4547-2215-5589-7865	02	24
2	Thomas Austin	784	98541	5214-2015-5632-4512	05	22
3	Keith Jam	124	95115	3564-1245-7841-8485	04	20
4	Gena Hopewell	168	93213	3539-1283-2029-5838	12	22
5	Bob Dang	985	98511	7454-1245-5457-6932	08	25
6	Cinderella McRitchie	212	93202	5002-3566-6226-9946	12	22
7	Hildagard Symington	651	95075	5610-4590-1040-5420	12	21
8	Eddy Ung	451	96235	7452-1254-4755-6542	01	27
9	Chiquia Dennington	643	91256	5183-0293-8061-3577	09	24
10	Kalindi Schubart	559	95653	3565-2575-2781-0847	07	23
11	Ronny Sa	541	97512	4564-2535-5667-2633	05	24
12	Odell McNaught	692	94068	3536-4509-0939-8434	01	21
13	Michele Sawl	623	92195	3567-3354-4948-4301	01	25
14	Aili Rean	273	91062	4211-9517-2088-6036	11	23
15	Heddi Esposita	518	91799	2018-8097-9437-1003	11	26
16	Loreen McIndoe	685	94621	3580-1560-6271-4789	12	24
17	Clea Allright	778	94068	2723-0115-9528-8936	06	22
18	Isaac Marlowe	685	92035	3038-1288-3039-8734	07	25
19	Alica Legh	303	94910	5038-9081-6431-0355	02	24
20	Tom Cruz	456	95132	4575-8956-4521-1254	11	27
21	Windy Storm	356	95111	1111-2222-3333-4444	02	25
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
5 •  SELECT * FROM Storefront.holds;
```

esult Grid | Filter Rows:

accountID	cardID
2	4
3	5
5	6
6	7
7	8
7	9
7	10
11	11
12	12
13	13
14	15
14	16
14	17
16	14
18	18
19	19
20	20
21	21
NULL	NULL

Code explanation:

- Frontend

```
export async function addCard(data) {
  fetch(`http://${url}/cards/add?id=${data.accountID}` +
    `&cardHolder=${data.cardHolder}&CVV=${data.CVV}` +
    `&Zip=${data.Zip}&CardNumber=${data.cardNumber}` +
    `&ExpMonth=${data.ExpMonth}&ExpYear=${data.ExpYear}`)
    .catch((err) => {
      console.error(err);
    });
}
```

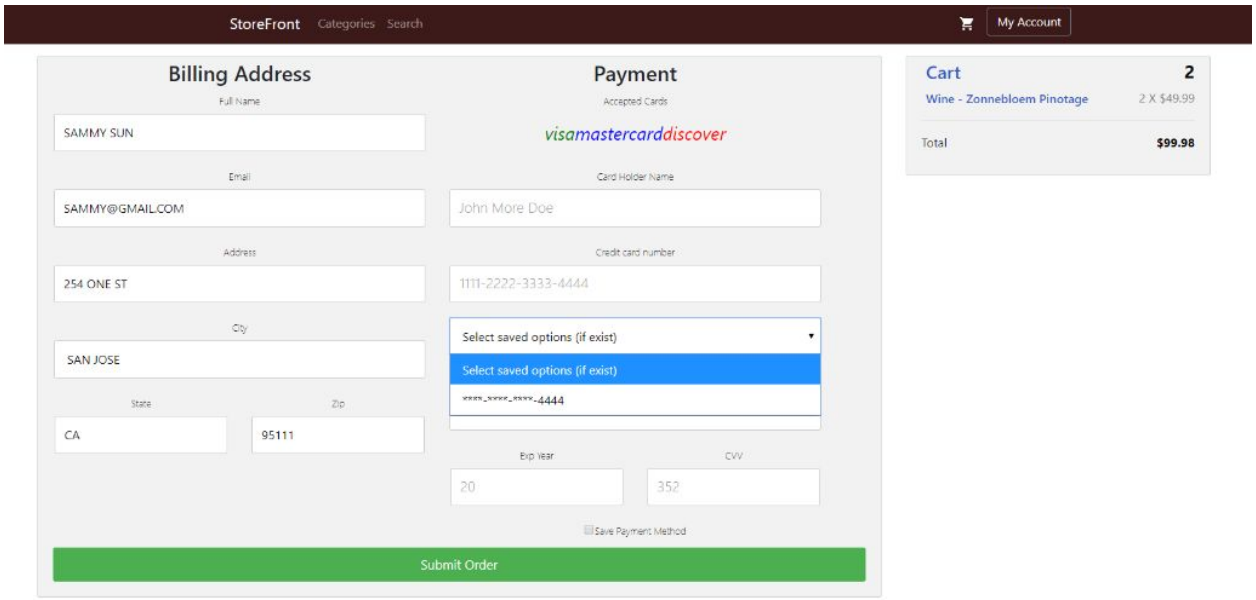
- Backend

```
app.get('/cards/add', (req, res) => {
    const { id, cardHolder, CVV, Zip, CardNumber, ExpMonth, ExpYear } = req.query;

    var lastInsert;
    pool.getConnection(function (err, con) {
        con.query(`insert into CardInfo(CardHolder, CVV, Zip, CardNumber, ExpMonth, ExpYear) values (
            '${cardHolder}', ${CVV}, ${Zip}, '${CardNumber}', '${ExpMonth}', '${ExpYear}'
        )`, (err, results) =>{
            if(err) res.send(err)
            else{
                lastInsert = results.insertId;
                con.query(`insert into holds values(${id}, ${lastInsert})`, (err, results) => {
                    if (err) res.send(err);
                    else res.send("Successfully added card to account");
                });
            }
        });
        con.release();
    });
});
```

  - Simply inserts the given information into the CardInfo table and then creates a relation to the accountID by storing the information into the holds table

View payment methods

User interface: the user can view all their payment methods by selecting the drop down menu that is located at the payment page



Database: it will get the information of the CardInfo table (we are looking at the card that belong to the accountID 21)

```
5 •  SELECT * FROM Storefront.CardInfo;
```

esult Grid | Filter Rows: | Edit: | Export/Import:

cardID	CardHolder	CVV	Zip	CardNumber	ExpMonth	ExpYear
1	Samuel Roger	845	974512	4547-2215-5589-7865	02	24
2	Thomas Austin	784	98541	5214-2015-5632-4512	05	22
3	Keith Jam	124	95115	3564-1245-7841-8485	04	20
4	Gena Hopewell	168	93213	3539-1283-2029-5838	12	22
5	Bob Dang	985	98511	7454-1245-5457-6932	08	25
6	Cinderella McRitchie	212	93202	5002-3566-6226-9946	12	22
7	Hildagard Symington	651	95075	5610-4590-1040-5420	12	21
8	Eddy Ung	451	96235	7452-1254-4755-6542	01	27
9	Chiquia Dennington	643	91256	5183-0293-8061-3577	09	24
10	Kalindi Schubart	559	95653	3565-2575-2781-0847	07	23
11	Ronny Sa	541	97512	4564-2535-5667-2633	05	24
12	Odell McNaught	692	94068	3536-4509-0939-8434	01	21
13	Michele Sawl	623	92195	3567-3354-4948-4301	01	25
14	Aili Rean	273	91062	4211-9517-2088-6036	11	23
15	Heddi Esposita	518	91799	2018-8097-9437-1003	11	26
16	Loreen McIndoe	685	94621	3580-1560-6271-4789	12	24
17	Clea Allright	778	94068	2723-0115-9528-8936	06	22
18	Isaac Marlowe	685	92035	3038-1288-3039-8734	07	25
19	Alica Legh	303	94910	5038-9081-6431-0355	02	24
20	Tom Cruz	456	95132	4575-8956-4521-1254	11	27
21	Windy Storm	356	95111	1111-2222-3333-4444	02	25
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
5 •  SELECT * FROM Storefront.holds;
```

esult Grid | Filter Rows:

accountID	cardID
2	4
3	5
5	6
6	7
7	8
7	9
7	10
11	11
12	12
13	13
14	15
14	16
14	17
16	14
18	18
19	19
20	20
21	21
NULL	NULL

Code explanation:

- Frontend

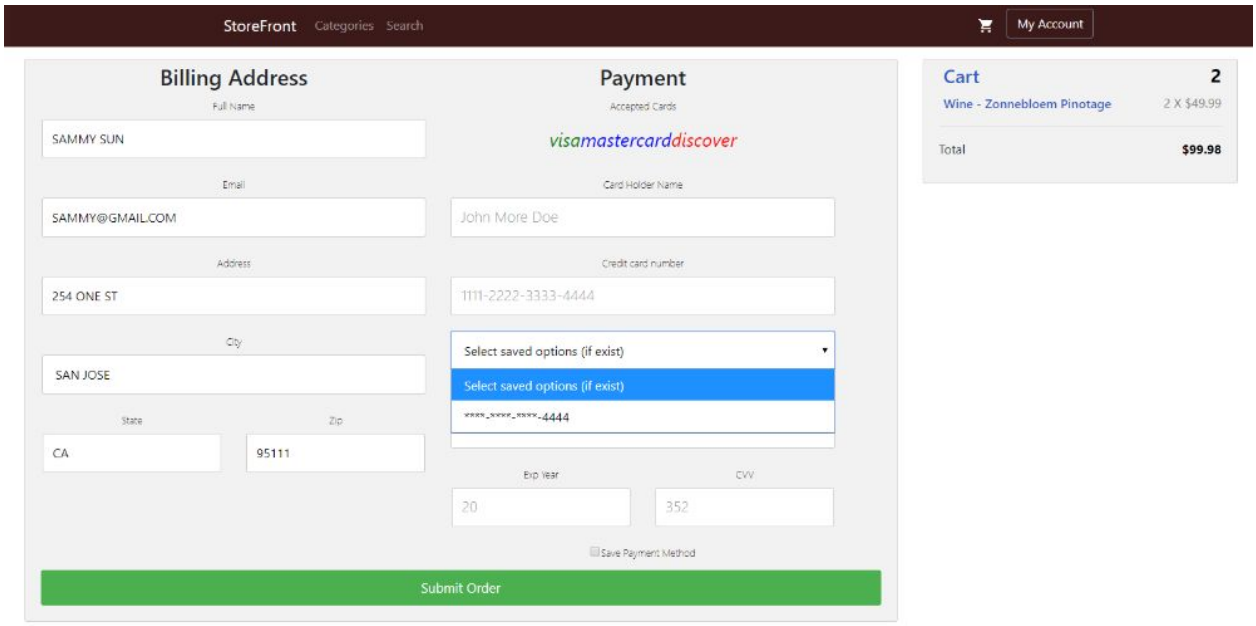
```
export async function getCards(user) {
  let objects = [];
  await fetch(`http://${url}/cards?id=${user.accountID}`)
    .then((response) => response.json())
    .then((response) => {
      objects = response;
    })
    .catch((err) => {
      console.error(err);
    });
  return objects;
}
```

- Backend
    - Returns all information associated with the given accountID

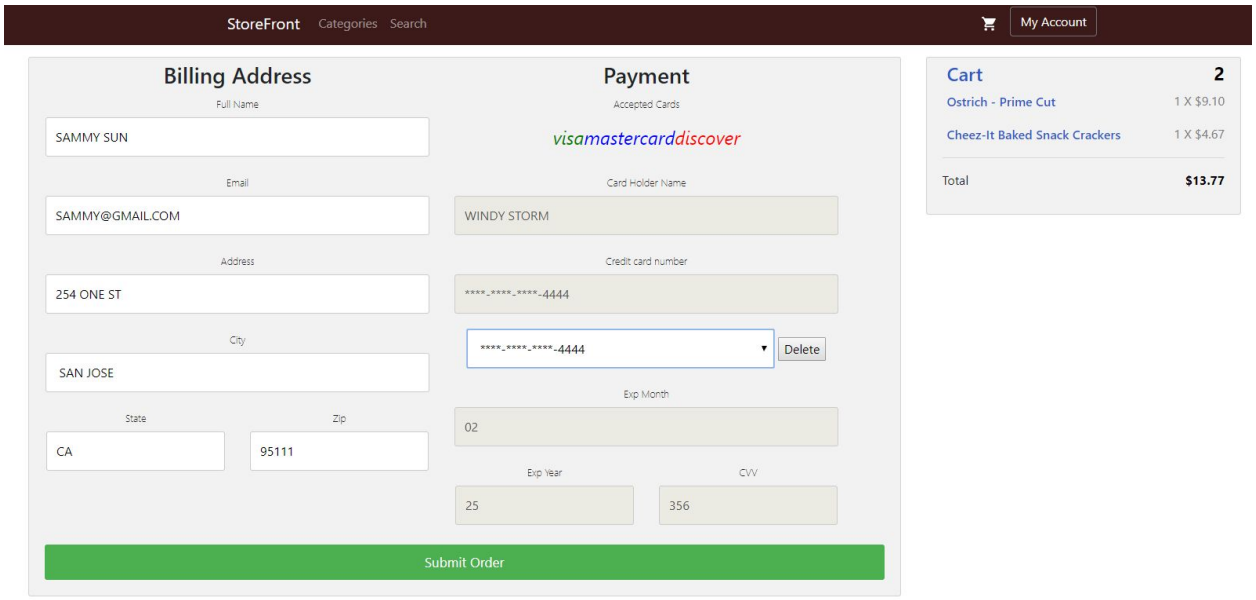
```
app.get('/cards', (req, res) => {
    const { id } = req.query;
    pool.getConnection(function (err, con) {
        con.query(`
        select
            cardID,
            CardHolder,
            CVV,
            Zip,
            CardNumber,
            ExpMonth,
            ExpYear
        from
            holds
        inner join
            CardInfo
        using
            (cardID)
        where
            accountID = ${ id}
        `, (err, results) => {
            if (err) res.send(err);
            else {
                res.send({
                    data: results
                });
            }
        });

        con.release();
    });
});
```

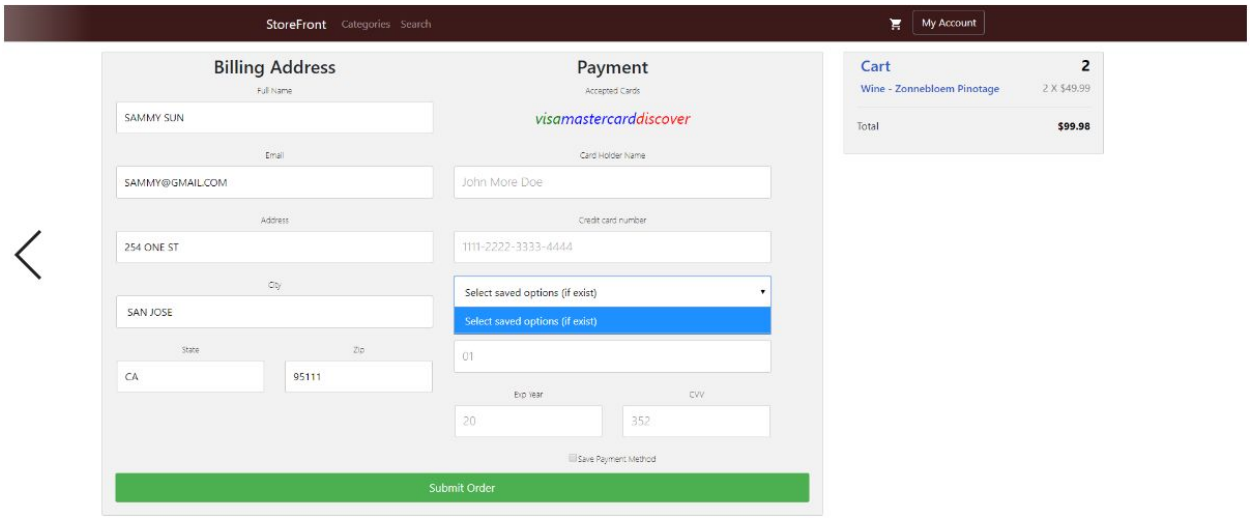
Deleting payment methods

User interface: Since there are the add new payment method, the user will also allow to add the payment that they don't want to or expired payment method that they don't want any more.



So for this example, we will try to delete the payment method ending in 4444





Database: it will get the information of the CardInfo table and holds relationships table (we are looking at the card that belong to the accountID 21)

Before delete a payment method:

```
5 •  SELECT * FROM Storefront.CardInfo;
```

esult Grid | Filter Rows: | Edit: | Export/Import:

cardID	CardHolder	CVV	Zip	CardNumber	ExpMonth	ExpYear
1	Samuel Roger	845	974512	4547-2215-5589-7865	02	24
2	Thomas Austin	784	98541	5214-2015-5632-4512	05	22
3	Keith Jam	124	95115	3564-1245-7841-8485	04	20
4	Gena Hopewell	168	93213	3539-1283-2029-5838	12	22
5	Bob Dang	985	98511	7454-1245-5457-6932	08	25
6	Cinderella McRitchie	212	93202	5002-3566-6226-9946	12	22
7	Hildagard Symington	651	95075	5610-4590-1040-5420	12	21
8	Eddy Ung	451	96235	7452-1254-4755-6542	01	27
9	Chiquia Dennington	643	91256	5183-0293-8061-3577	09	24
10	Kalindi Schubart	559	95653	3565-2575-2781-0847	07	23
11	Ronny Sa	541	97512	4564-2535-5667-2633	05	24
12	Odell McNaught	692	94068	3536-4509-0939-8434	01	21
13	Michele Sawl	623	92195	3567-3354-4948-4301	01	25
14	Aili Rean	273	91062	4211-9517-2088-6036	11	23
15	Heddi Esposita	518	91799	2018-8097-9437-1003	11	26
16	Loreen McIndoe	685	94621	3580-1560-6271-4789	12	24
17	Clea Allright	778	94068	2723-0115-9528-8936	06	22
18	Isaac Marlowe	685	92035	3038-1288-3039-8734	07	25
19	Alica Legh	303	94910	5038-9081-6431-0355	02	24
20	Tom Cruz	456	95132	4575-8956-4521-1254	11	27
21	Windy Storm	356	95111	1111-2222-3333-4444	02	25
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
5 •  SELECT * FROM Storefront.holds;
```

esult Grid | Filter Rows:

accountID	cardID
2	4
3	5
5	6
6	7
7	8
7	9
7	10
11	11
12	12
13	13
14	15
14	16
14	17
16	14
18	18
19	19
20	20
21	21
NULL	NULL

After a payment card is deleted, in this case, the cardID 21 does not exist in the database anymore

```
5 •  SELECT * FROM Storefront.CardInfo;
```

esult Grid | Filter Rows: | Edit: | Export/Import:

cardID	CardHolder	CVV	Zip	CardNumber	ExpMonth	ExpYear
1	Samuel Roger	845	974512	4547-2215-5589-7865	02	24
2	Thomas Austin	784	98541	5214-2015-5632-4512	05	22
3	Keith Jam	124	95115	3564-1245-7841-8485	04	20
4	Gena Hopewell	168	93213	3539-1283-2029-5838	12	22
5	Bob Dang	985	98511	7454-1245-5457-6932	08	25
6	Cinderella McRitchie	212	93202	5002-3566-6226-9946	12	22
7	Hildagard Symington	651	95075	5610-4590-1040-5420	12	21
8	Eddy Ung	451	96235	7452-1254-4755-6542	01	27
9	Chiquia Dennington	643	91256	5183-0293-8061-3577	09	24
10	Kalindi Schubart	559	95653	3565-2575-2781-0847	07	23
11	Ronny Sa	541	97512	4564-2535-5667-2633	05	24
12	Odell McNaught	692	94068	3536-4509-0939-8434	01	21
13	Michele Sawl	623	92195	3567-3354-4948-4301	01	25
14	Aili Rean	273	91062	4211-9517-2088-6036	11	23
15	Heddi Esposita	518	91799	2018-8097-9437-1003	11	26
16	Loreen McIndoe	685	94621	3580-1560-6271-4789	12	24
17	Clea Allright	778	94068	2723-0115-9528-8936	06	22
18	Isaac Marlowe	685	92035	3038-1288-3039-8734	07	25
19	Alica Legh	303	94910	5038-9081-6431-0355	02	24
20	Tom Cruz	456	95132	4575-8956-4521-1254	11	27
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
5 •  SELECT * FROM Storefront.holds;
```

Result Grid | Filter Rows:

accountID	cardID
2	4
3	5
5	6
6	7
7	8
7	9
7	10
11	11
12	12
13	13
14	15
14	16
14	17
16	14
18	18
19	19
20	20
NULL	NULL

Code explanation:

- Frontend

```
export async function deleteCard(data) {
  fetch(`http://${url}/cards/remove?id=${data.accountID}&cardID=${data.cardID}`)
    .catch((err) => {
      console.error(err);
    });
}
```

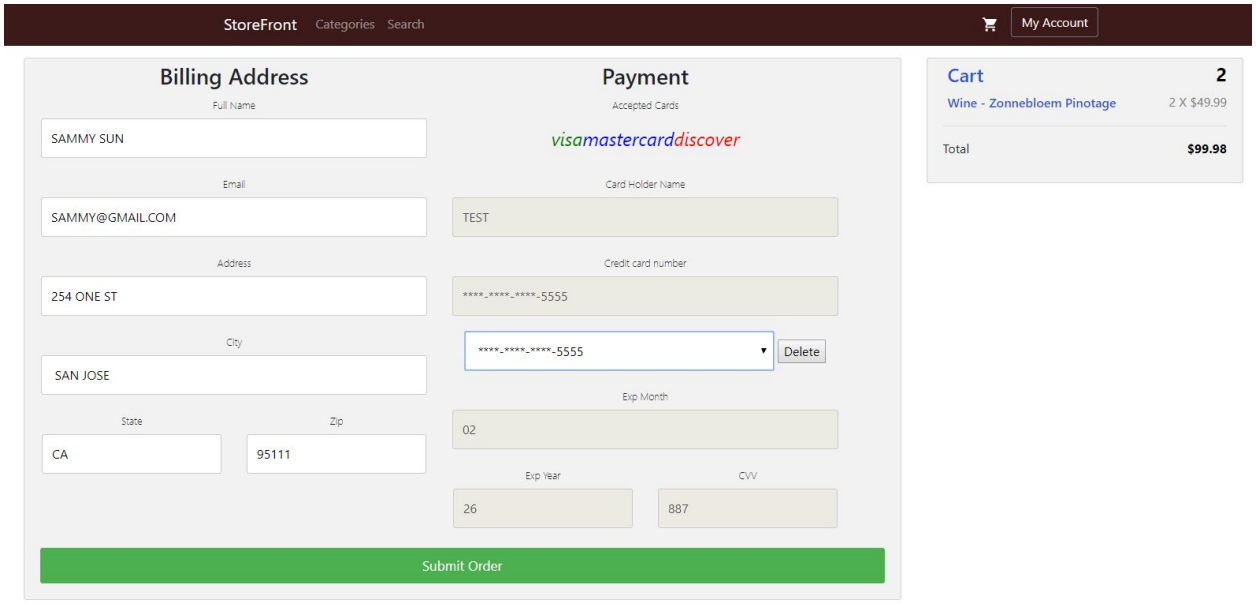
- Backend

```
app.get('/cards/remove', (req, res) => {
    const { id, cardID } = req.query;
    pool.getConnection(function (err, con) {
        con.query(`select * from holds where accountID = ${id} and cardID = ${cardID}`, (err, results) => {
            if (err) res.send(err);
            else {
                if (results.length == 1) {
                    con.query(`delete from CardInfo where cardID = ${cardID}`, (err, results) => {
                        if (err) res.send(err);
                        else {
                            con.query(`delete from holds where accountID = ${id} and cardID = ${cardID}`, (err, results) => {
                                if (err) res.send(err);
                                else res.send(`Successfully deleted card of id ${cardID}`);
                            });
                        }
                    });
                } else {
                    res.send(`Card of ID ${cardID} not found`)
                }
            }
        });
        con.release();
    });
});
```

  - Deletes the Card of specified ID and dissociates it with the given accountID

Purchasing items

User Interface: after the user is satisfied with the items that are/is in their cart, they can go to the payment method, fill in all the information and then click on the “Submit Order” and they are done.



Database: when the order is made, there are a few things that are happening in the background.

First, the CartInfo table will get cleared

Before new order is make

```
SELECT * FROM Storefront.Cart;
```

Filter Rows:

accountID	itemID	quantity
1	1	1
1	2	1
1	4	3
2	1	1
5	1	2
5	6	1
6	7	3
6	9	1
6	10	1
7	8	1
7	9	2
8	1	1
8	9	2
10	2	1
10	3	2
10	5	1
10	6	1
21	15	2
NULL	NULL	NULL

After new order is made

```
SELECT * FROM Storefront.Cart;
```

Filter Rows:

accountID	itemID	quantity
1	1	1
1	2	1
1	4	3
2	1	1
5	1	2
5	6	1
6	7	3
6	9	1
6	10	1
7	8	1
7	9	2
8	1	1
8	9	2
10	2	1
10	3	2
10	5	1
10	6	1
NULL	NULL	NULL

Second, the Order table, contain relationship Table, and make relationship Table get modified

Before a new order is made:

```
5 •  SELECT * FROM Storefront.make;
```

esult Grid | Filter Rows:

accountID	orderID
1	1
1	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15

```
5 •  SELECT * FROM Storefront.contain;
```

esult Grid | Filter Rows: | Edit:

orderID	itemID	quantity
1	1	2
1	2	3
2	2	2
3	3	1
4	4	1
5	5	4
6	6	1
7	7	2
8	8	3
9	9	1
10	11	5
11	11	1
12	12	1
13	13	3
14	14	4
15	15	1

```
5 •  SELECT * FROM Storefront.Orders;
```

esult Grid | Filter Rows:

orderID	price
1	53.1
2	18.2
3	9.1
4	6.3
5	22.5
6	23.99
7	6.2
8	9.9
9	7.1
10	75.5
11	15.1
12	4.75
13	15.3
14	102.16
15	49.99

After the new order is made:

```
5 •  SELECT * FROM Storefront.make;
```

esult Grid | Filter Rows:

accountID	orderID
1	1
1	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
21	73

```
5 •  SELECT * FROM Storefront.contain;
```

esult Grid | Filter Rows: | Edit

orderID	itemID	quantity
1	1	2
1	2	3
2	2	2
3	3	1
4	4	1
5	5	4
6	6	1
7	7	2
8	8	3
9	9	1
10	11	5
11	11	1
12	12	1
13	13	3
14	14	4
15	15	1
73	15	2
NULL	NULL	NULL

```
5 •  SELECT * FROM Storefront.Orders;
```

esult Grid | Filter Rows: | E

orderID	price
1	53.1
2	18.2
3	9.1
4	6.3
5	22.5
6	23.99
7	6.2
8	9.9
9	7.1
10	75.5
11	15.1
12	4.75
13	15.3
14	102.16
15	49.99
73	99.98

Code explanation:

- Frontend

```
export async function addOrders(data, accountID) {
  let dataString = JSON.stringify(data);
  let fetchString = `http://${url}/orders/add?items=` +
    dataString + "&accountID=" + accountID;

  fetch(fetchString)
    .catch((err) => {
      console.error(err);
    });
}
```

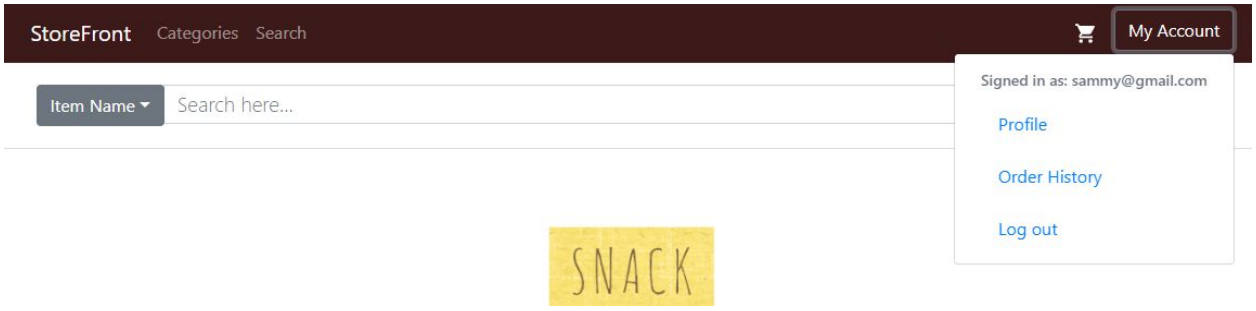
- Backend

```
app.get(`/orders/add`, (req, res) => {
    var accountID = req.query.accountID;
    var items = JSON.parse(req.query.items);
    var prices = 0;
    var lastInsert;
    var query = `select price from Item where`;

    for (i = 0; i < items.length; i++) {
        if (i == 0) query += ` itemID=${items[0][0]}`
        else query += ` or itemID=${items[i][0]}`
    }

    pool.getConnection(function (err, con) {
        con.query(`insert into Orders(price) values('0')`, (err, results) => {
            if (err) res.sendStatus(500);
            else {
                lastInsert = results.insertId;
                for (i = 0; i < items.length; i++) {
                    con.query(`insert into contain(orderID, itemID, quantity) values(${lastInsert},${items[i][0]},${items[i][1]})`, (err, results) => {
                        if (err) res.sendStatus(500);
                    });
                }
                con.query(query, (err, results) => {
                    if (err) res.sendStatus(500);
                    else {
                        for (i = 0; i < results.length; i++) {
                            prices += results[i].price * items[i][1];
                        }
                        con.query(`update Orders set price='${prices}' where orderID='${lastInsert}'`, (err, results) => {
                            if (err) res.sendStatus(500);
                            else {
                                con.query(`insert into make(accountID, orderID) values('${accountID}', '${lastInsert}')`, (err, results) => {
                                    if (err) res.sendStatus(500);
                                    else {
                                        con.query(`delete from Cart where accountID=${accountID}`, (err, results) => {
                                            if (err) res.sendStatus(500);
                                            else res.send(`Successfully created order with id ${lastInsert} associated with account ${accountID}`);
                                        })
                                    }
                                });

                            }
                        });

                    }
                });
            }
        });

        con.release();
    });
});
```

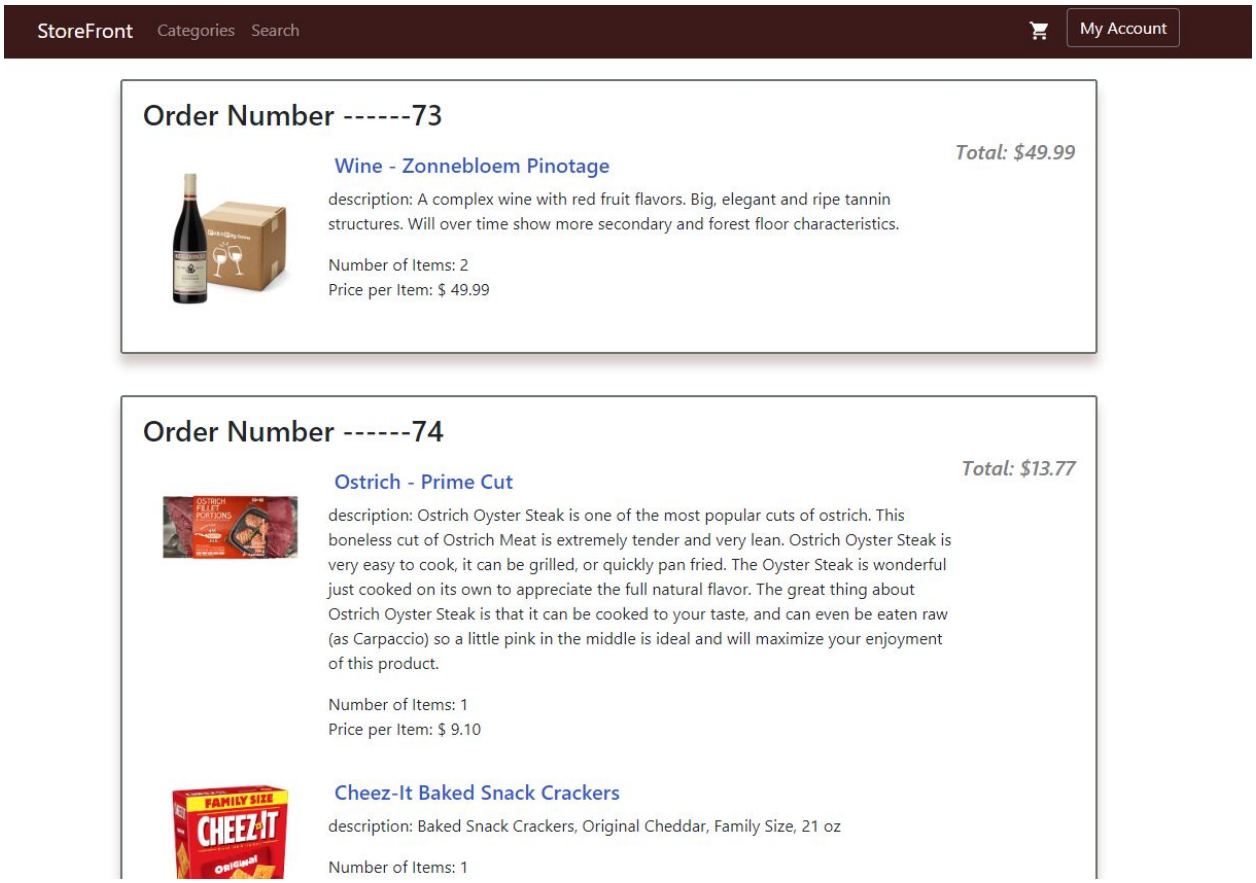
    - Adds an item into the orders table which stores order history. First creates a tuple that holds base values. Then creates a relation between that tuple and the accountID by inserting into the contain table. After that it updates the tuple in Orders to the correct price and then deletes the Cart.

View Order History

User Interface: the user can always go to their order history by clicking on “My Account” tab located at the right of the navigation bar. Then the “Order History” option will show up.



By clicking on it, they will able to see all of the orders that they had placed



Database: the query of this will involving the Order, make, and contain Tables

```
5 •  SELECT * FROM Storefront.make;
```

esult Grid | Filter Rows:

accountID	orderID
1	1
1	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
21	73
21	74
NULL	NULL

```
5 •  SELECT * FROM Storefront.contain;
```

esult Grid | Filter Rows: | Ed

orderID	itemID	quantity
1	1	2
1	2	3
2	2	2
3	3	1
4	4	1
5	5	4
6	6	1
7	7	2
8	8	3
9	9	1
10	11	5
11	11	1
12	12	1
13	13	3
14	14	4
15	15	1
73	15	2
74	3	1
74	16	1
NULL	NULL	NULL

```
5 •  SELECT * FROM Storefront.Orders;
```

esult Grid | Filter Rows: | Ed

orderID	price
1	53.1
2	18.2
3	9.1
4	6.3
5	22.5
6	23.99
7	6.2
8	9.9
9	7.1
10	75.5
11	15.1
12	4.75
13	15.3
14	102.16
15	49.99
73	99.98
74	13.77
NULL	NULL

Code explanation:

- Frontend

```
export async function getOrders(accountID, orderID) {
  let objects = [];
  const query = !orderID ?
    `http://${url}/orders?id=${accountID}` :
    `http://${url}/orders?id=${accountID}&orderID=${orderID}`;

  await fetch(`${query}`)
    .then((response) => response.json())
    .then((response) => {
      objects = response;
    })
    .catch((err) => {
      console.error(err);
    });
  return objects;
}
```

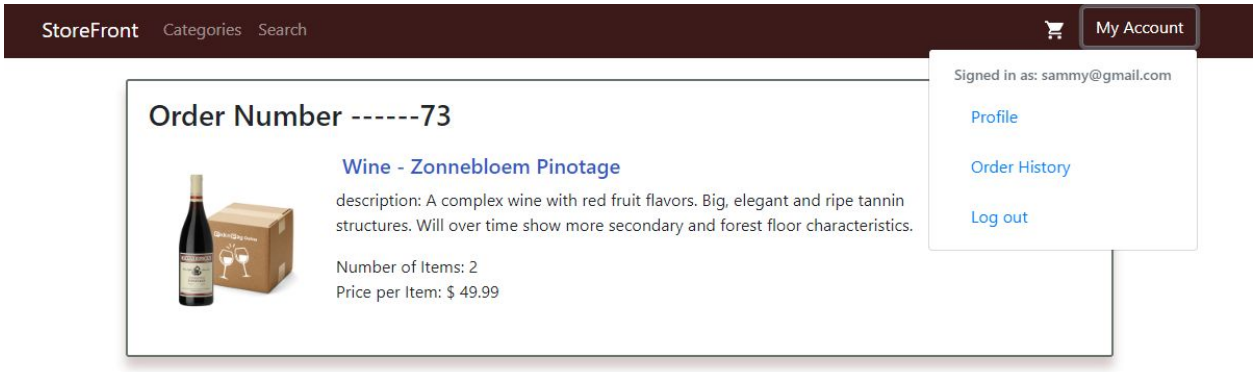
- Backend

```
app.get('/orders', (req, res) => {
    const { id, orderID } = req.query;
    pool.getConnection(function (err, con) {
        if (!orderID) {
            con.query(`
            select orderID, price, itemID, quantity, itemName, itemPrice,
                image, description
            from Orders natural join make natural join contain
            natural join
            (select itemID, itemName, price as itemPrice,
                description, image from Item) i
            where accountID = ${id}
            `, (err, results) => {
                if (err) res.send(err);
                else res.send(results);
            });
        } else {
            con.query(`
            select orderID, price, itemID, quantity, itemName, itemPrice,
                image, description
            from Orders natural join make natural join contain
            natural join
            (select itemID, itemName, price as itemPrice,
                description, image from Item) i
            where accountID = ${id} and orderID = ${orderID}
            `, (err, results) => {
                if (err) res.send(err);
                else res.send(results);
            });
        }

        con.release();
    });
});
```

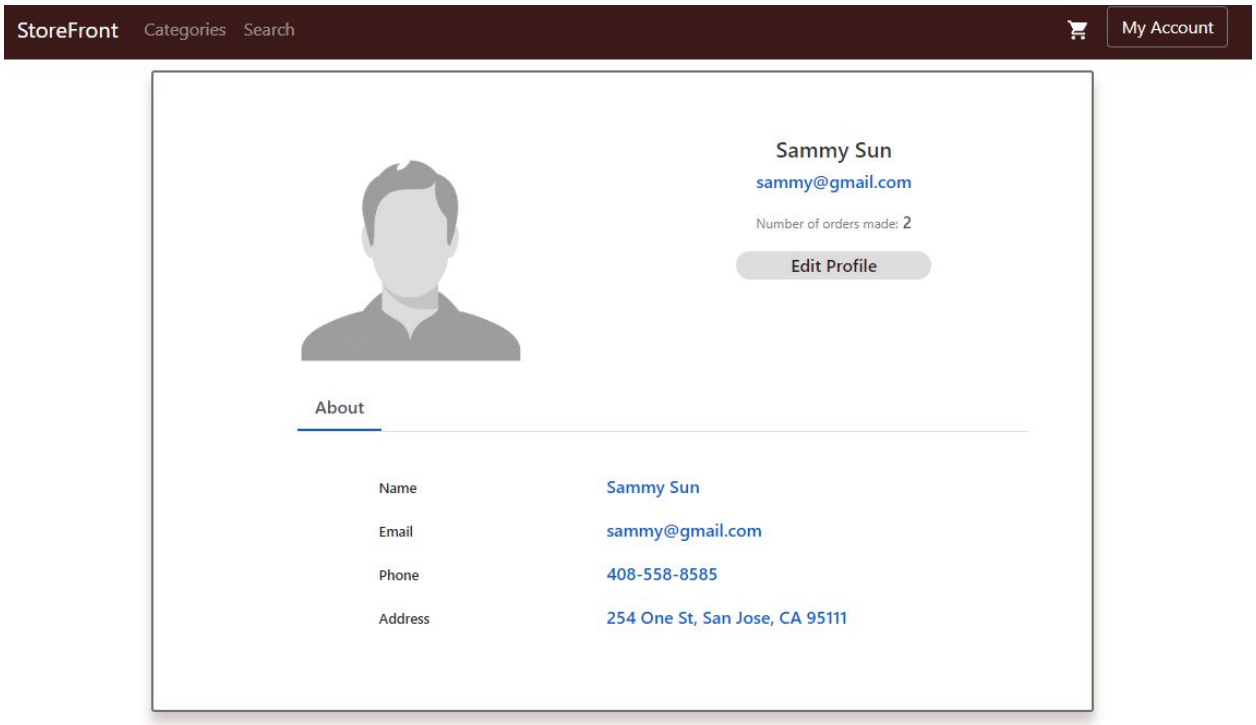
    - If no orderID is sent then it will return all the orders associated with the given accountID. Otherwise it will return the specific order that is given.

View account information and email

User Interface: Similar to the View Order History, the user can view their information by clicking on “My Account” tab located at the right of the navigation bar. Then the “Profile” option will show up.



Then they should see something like this



Database: this will just require a simple query that get all the information in the Account table (note: the accountID we use for this example is 21)

```
5 •  SELECT * FROM Storefront.Account;
```

esult Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

accountID	email	password	name	cell	address	sessionID
1	ascarsbrick0@blogspot.com	f651472f05ed929bab24b7da3a92d207	Albina Scarsbrick	273-816-3890	08106 Golf View Street	145fr-545-f4r-45
2	scrocket1@themeforest.net	21ec4b2edb2260da03aebb2a95fbcdlca	Sam Crocket	202-468-7307	7752 Pine View Circle	a14-7845-f5d-554r
3	nbasili2@alexa.com	3da13526855124534338fb5154668cc7	Nolie Basili	994-128-9298	334 Melody Plaza	54bo-45li5-85
4	gpietruszewicz3@bbb.org	d6acc12d4d2bd2839fe6fc4b828165e	Gail Pietruszewic	269-647-6574	7 Green Ridge Street, Santa Clara, CA 95842	jk-7841-hy55
5	rbend4@webnode.com	bba8cf9d0c4b1f7b099b14cbd4bccf79	Row Bend	747-339-2047	727 Dunning Court	125g-jk2-552
6	tpoundford5@wikispaces.com	eb7140ec45b0jca2be9a8c4a89f98197d	Terrel Poundford	708-331-7934	829 Rowland Trail	m12g5-452g
7	earnal6@quantcast.com	5963ff246cdb21b59d4f1a0f5ba3e000	Erna Arnal	983-528-1119	3009 La Follette Circle	7bg-8952-fbs87
8	lcharkham7@networksolutions.com	fd3145fe4c72570d0acf160004d95a1d	Liza Charkham	683-846-4023	50572 Cherokee Terrace	1357g8-mhj5-897
9	fmusslewhite8@timesonline.co.uk	8fb08a6d9197b4ed375116d6f6807f1a	Felipa Musslewhite	340-152-1797	8 Stephen Pass	a1d4-785b-794
10	ilemmon9@pcworld.com	736973eb31ddb430801d72e125eb1434	Innis Lemmon	991-223-7323	17 Toban Plaza	m1g4-85g-545-fh45
11	mtriplowa@google.co.jp	928ea5b041a58ec5810d8e24eab2887f	Mady Triplow	295-443-8330	8270 Green Crossing	5o45u-78o545-l2
12	rshadrackb@zimbio.com	82eaf1fb7d4dd80492393b5d035123c8	Reuben Shadrack	868-906-4924	5623 Victoria Drive	124f-452-k25-l45
13	fannettsc@fc2.com	a228ce69cb8a69afeb841507d238cdc3	Freeman Annetts	879-130-2731	921 Heath Pass	ht-45f-t4g-ff4
14	ikybertd@pagesperso-orange.fr	6abfa1dc3e8b0602486bea8b7731f9b5	Isac Kybert	370-449-1348	072 Luster Center	fg-784h-45fr-4b
15	tjerwoode@google.nl	de73719f0e26769f2cdaa99a8af89fe5	Teddie Jerwood	803-660-0104	199 Rowland Drive	87d-ht-454-bl4
16	jbinningf@naver.com	cee4937e8dd933d7a0ff6f7f0a1794a1	Justina Binning	151-469-1415	815 Dovetail Alley	1bl5-525-8f5dg-58...
17	lharesnapeg@shinystat.com	5d3e2927dd5ddddd521659cb7cbc3436	Lise Haresnape	925-304-2484	208 Bunker Hill Road	8411-787d-44f-45gd
18	nbyeh@ovh.net	99e9e9af8bc7619c747ed26f32289dde	Ninon Bye	521-582-2732	797 Eliot Drive	454eb-d554-f5d-r4...
19	cmonceyi@w3.org	c1bc8fb1d078142cffaa0e9818b6aa6a	Charles Moncey	943-403-6754	56 Mccormick Terrace	ef451-895s-45fs-5...
20	aastlej@mysql.com	81b4d83856aa1beff8eeb34bb23bd841	Adoree Astle	406-383-4369	2 Katie Plaza	215e-sv54-s5852
21	sammy@gmail.com	fab459fa5a45eb2402f7cef48ca630b8	Sammy Sun	408-558-8585	254 One St, San Jose, CA 95111	3e5bf0f1-f08b-5ed...
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Code explanation:

- Frontend

```
export async function getUser(user) {
  let objects = [];
  await fetch(`http://${url}/users?id=${user.accountID}`)
    .then((response) => response.json())
    .then((response) => {
      objects = response;
    })
    .catch((err) => {
      console.error(err);
    });
  return objects;
}
```

- Backend

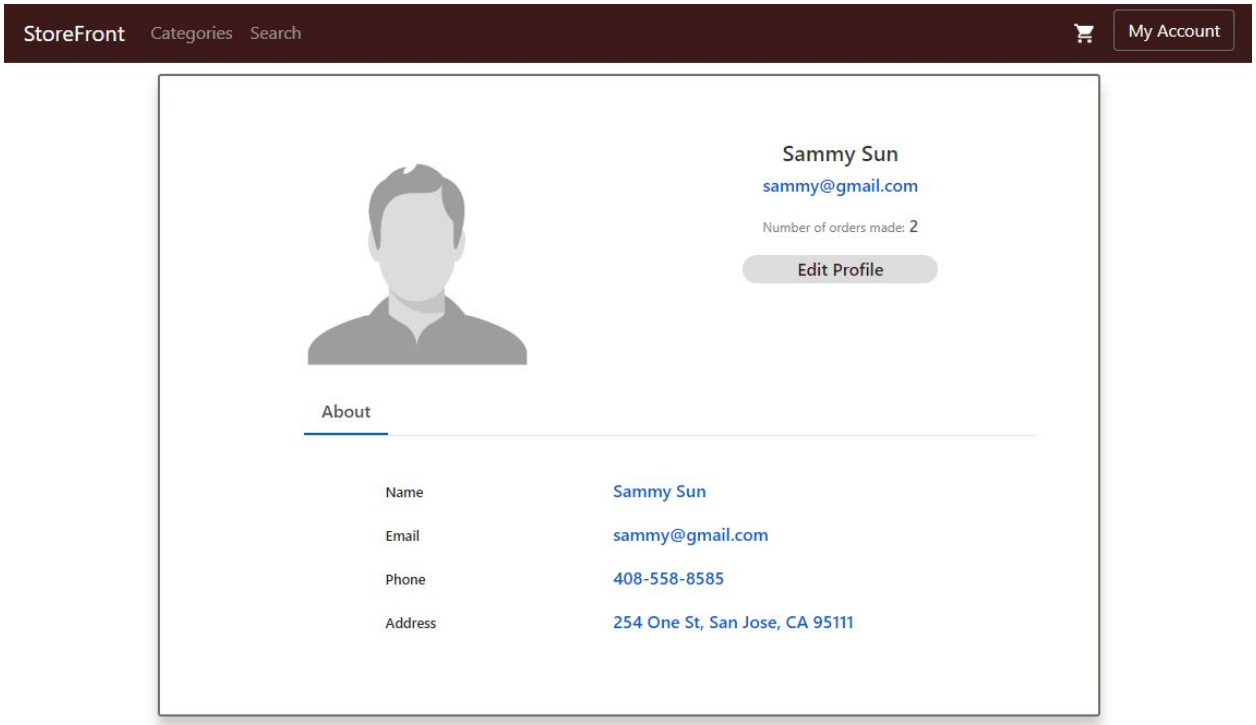
```
app.get('/users', (req, res) => {
    const { id } = req.query;
    pool.getConnection(function (err, con) {
        con.query(`select * from Account where accountID=${id}`, (err, results) => {
            if (err) res.send(err);
            else {
                res.send({
                    ...results
                });
            }
        });
        con.release();
    });
});
```

  - Returns all tuples associated with the given accountID

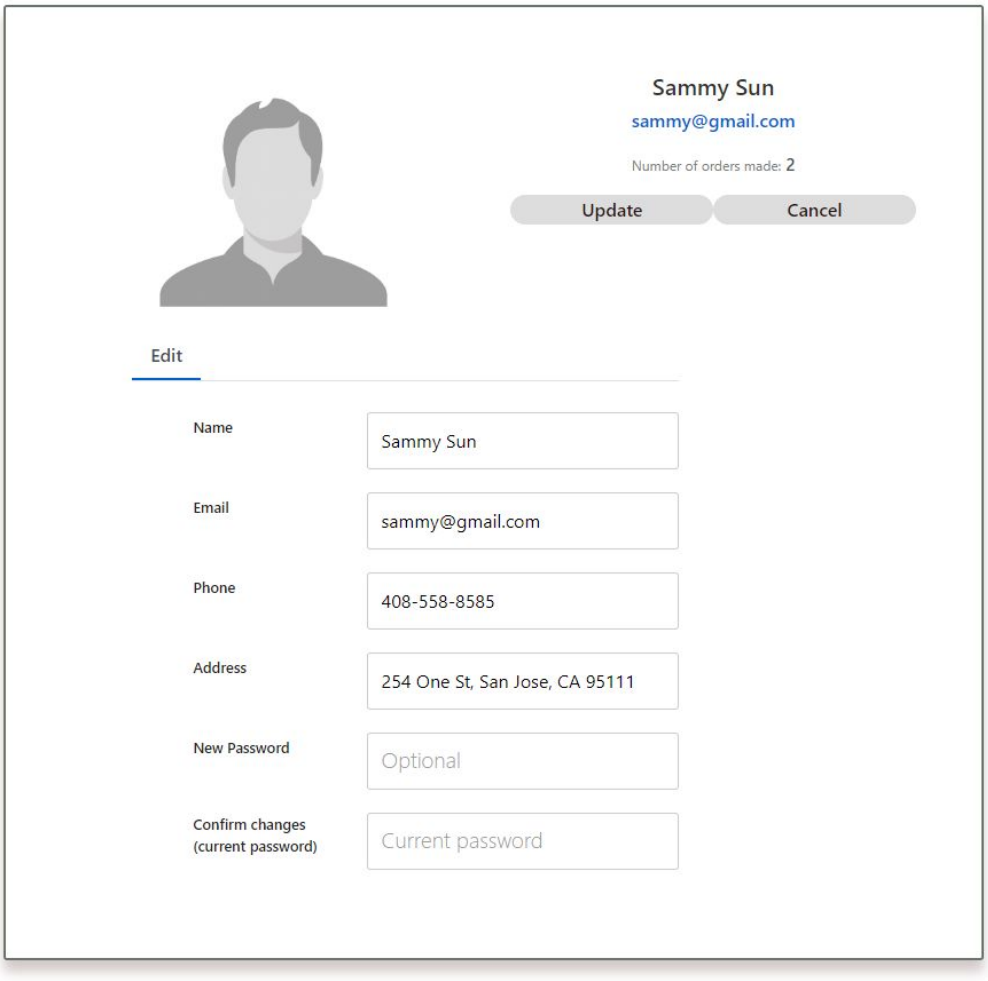
Changing email and password

User Interface: Just like other usual web application, the user will also be allowed to edit their profile information.

To do this, all they have to do is click on the “Edit Profile” button under the Profile page



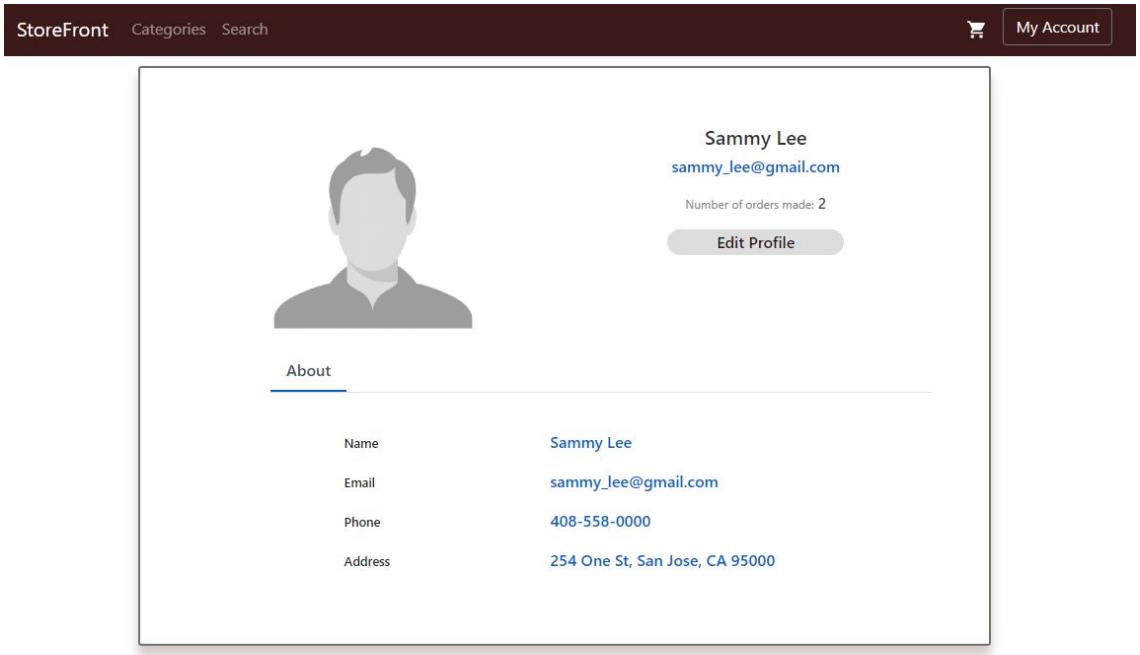
What they can see is this:



As you can see in the picture above, the name, email, phone and address are autofill for the user. All they have to do is change any information that they want. After the user is satisfied with the changes, they will be required to enter their current password for a confirmation. (note that the password will not change unless the user enters something inside the new password field).

For this example, lets change user's name from "Sammy Sun" to "Sammy Lee", email from "sammy@gmail.com" to "sammy_lee@gmail.com", phone number from "408-558-8585" to "408-558-0000", and address from "254 One St, San Jose, CA 95111" to "254 One St, San Jose, CA 95000".


Sammy Sun
sammy@gmail.com
Number of orders made: 2
Update
Cancel
Edit
Name
Sammy Lee
Email
sammy_lee@gmail.com
Phone
408-558-0000
Address
254 One St, San Jose, CA 95000
New Password
Optional
Confirm changes
(current password)
••••••••


StoreFront
Categories
Search
My Account
Sammy Lee
sammy_lee@gmail.com
Number of orders made: 2
Edit Profile
About
Name
Sammy Lee
Email
sammy_lee@gmail.com
Phone
408-558-0000
Address
254 One St, San Jose, CA 95000

Database: in the background, there will be a query that will modify the information of the Account table based on the accountID (note: we will look at accountID 21 for this example)

Before profile information is changes:

```
5 •   SELECT * FROM Storefront.Account;
```

esult Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

accountID	email	password	name	cell	address	sessionID
1	ascarsbrick0@blogspot.com	f651472f05ed929bab24b7da3a92d207	Albina Scarsbrick	273-816-3890	08106 Golf View Street	145fr-545-f4r-45
2	scrocket1@themeforest.net	21ec4b2edb2260da03aebb2a95fbcdlca	Sam Crocket	202-468-7307	7752 Pine View Circle	a14-7845-f5d-554r
3	nbasili2@alexa.com	3da13526855124534338fb5154668cc7	Nolie Basili	994-128-9298	334 Melody Plaza	54bo-45li5-85
4	gpietruszewicz3@bbb.org	d6acc12d4d2bd2839fe6fc4b828165e	Gail Pietruszewic	269-647-6574	7 Green Ridge Street, Santa Clara, CA 95842	jk-7841-hy55
5	rbend4@webnode.com	bba8cf9d0c4b1f7b099b14cbd4bccf79	Row Bend	747-339-2047	727 Dunning Court	125g-jk2-552
6	tpoundford5@wikispaces.com	eb7140ec45b0jca2be9a8c4a89f98197d	Terrel Poundford	708-331-7934	829 Rowland Trail	m12g5-452g
7	earnal6@quantcast.com	5963ff246cdb21b59d4f1a0f5ba3e000	Erna Arnal	983-528-1119	3009 La Follette Circle	7bg-8952-fbs87
8	lcharkham7@networksolutions.com	fd3145fe4c72570d0acf160004d95a1d	Liza Charkham	683-846-4023	50572 Cherokee Terrace	1357g8-mhj5-897
9	fmusslewhite8@timesonline.co.uk	8fb08a6d9197b4ed375116d6f6807f1a	Felipa Musslewhite	340-152-1797	8 Stephen Pass	a1d4-785b-794
10	ilemmon9@pcworld.com	736973eb31ddb430801d72e125eb1434	Innis Lemmon	991-223-7323	17 Toban Plaza	m1g4-85g-545-fh45
11	mtriplowa@google.co.jp	928ea5b041a58ec5810d8e24eab2887f	Mady Triplow	295-443-8330	8270 Green Crossing	5o45u-78o545-l2
12	rshadrackb@zimbio.com	82eaf1fb7d4dd80492393b5d035123c8	Reuben Shadrack	868-906-4924	5623 Victoria Drive	124f-452-k25-l45
13	fannettsc@fc2.com	a228ce69cb8a69afeb841507d238cdc3	Freeman Annetts	879-130-2731	921 Heath Pass	ht-45f-t4g-ff4
14	ikybertd@pagesperso-orange.fr	6abfa1dc3e8b0602486bea8b7731f9b5	Isac Kybert	370-449-1348	072 Luster Center	fg-784h-45fr-4b
15	tjerwoode@google.nl	de73719f0e26769f2cdaa99a8af89fe5	Teddie Jerwood	803-660-0104	199 Rowland Drive	87d-ht-454-bl4
16	jbinningf@naver.com	cee4937e8dd933d7a0ff6f7f0a1794a1	Justina Binning	151-469-1415	815 Dovetail Alley	1bl5-525-8f5dg-58...
17	lharesnapeg@shinystat.com	5d3e2927dd5ddddd521659cb7cbc3436	Lise Haresnape	925-304-2484	208 Bunker Hill Road	8411-787d-44f-45gd
18	nbyeh@ovh.net	99e9e9af8bc7619c747ed26f32289dde	Ninon Bye	521-582-2732	797 Eliot Drive	454eb-d554-f5d-r4...
19	cmonceyi@w3.org	c1bc8fb1d078142cffaa0e9818b6aa6a	Charles Moncey	943-403-6754	56 Mccormick Terrace	ef451-895s-45fs-5...
20	aastlej@mysql.com	81b4d83856aa1beff8eeb34bb23bd841	Adoree Astle	406-383-4369	2 Katie Plaza	215e-sv54-s5852
21	sammy@gmail.com	fab459fa5a45eb2402f7cef48ca630b8	Sammy Sun	408-558-8585	254 One St, San Jose, CA 95111	3e5bf0f1-f08b-5ed...
NULL	NULL	NULL	NULL	NULL	NULL	NULL

After the profile information is changed:

```
5 •   SELECT * FROM Storefront.Account;
```

esult Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

accountID	email	password	name	cell	address	sessionID
1	ascarsbrick0@blogspot.com	f651472f05ed929bab24b7da3a92d207	Albina Scarsbrick	273-816-3890	08106 Golf View Street	145fr-545-f4r-45
2	scrocket1@themeforest.net	21ec4b2edb2260da03aebb2a95fbcdlca	Sam Crocket	202-468-7307	7752 Pine View Circle	a14-7845-f5d-554r
3	nbasili2@alexa.com	3da13526855124534338fb5154668cc7	Nolie Basili	994-128-9298	334 Melody Plaza	54bo-45li5-85
4	gpietruszewicz3@bbb.org	d6acc12d4d2bd2839fe6fc4b828165e	Gail Pietruszewic	269-647-6574	7 Green Ridge Street, Santa Clara, CA 95842	jk-7841-hy55
5	rbend4@webnode.com	bba8cf9d0c4b1f7b099b14cbd4bccf79	Row Bend	747-339-2047	727 Dunning Court	125g-jk2-552
6	tpoundford5@wikispaces.com	eb7140ec45b0jca2be9a8c4a89f98197d	Terrel Poundford	708-331-7934	829 Rowland Trail	m12g5-452g
7	earnal6@quantcast.com	5963ff246cdb21b59d4f1a0f5ba3e000	Erna Arnal	983-528-1119	3009 La Follette Circle	7bg-8952-fbs87
8	lcharkham7@networksolutions.com	fd3145fe4c72570d0acf160004d95a1d	Liza Charkham	683-846-4023	50572 Cherokee Terrace	1357g8-mhj5-897
9	fmusslewhite8@timesonline.co.uk	8fb08a6d9197b4ed375116d6f6807f1a	Felipa Musslewhite	340-152-1797	8 Stephen Pass	a1d4-785b-794
10	ilemmon9@pcworld.com	736973eb31ddb430801d72e125eb1434	Innis Lemmon	991-223-7323	17 Toban Plaza	m1g4-85g-545-fh45
11	mtriplowa@google.co.jp	928ea5b041a58ec5810d8e24eab2887f	Mady Triplow	295-443-8330	8270 Green Crossing	5o45u-78o545-l2
12	rshadrackb@zimbio.com	82eaf1fb7d4dd80492393b5d035123c8	Reuben Shadrack	868-906-4924	5623 Victoria Drive	124f-452-k25-l45
13	fannettsc@fc2.com	a228ce69cb8a69afeb841507d238cdc3	Freeman Annetts	879-130-2731	921 Heath Pass	ht-45f-t4g-ff4
14	ikybertd@pagesperso-orange.fr	6abfa1dc3e8b0602486bea8b7731f9b5	Isac Kybert	370-449-1348	072 Luster Center	fg-784h-45fr-4b
15	tjerwoode@google.nl	de73719f0e26769f2cdaa99a8af89fe5	Teddie Jerwood	803-660-0104	199 Rowland Drive	87d-ht-454-bl4
16	jbinningf@naver.com	cee4937e8dd933d7a0ff6f7f0a1794a1	Justina Binning	151-469-1415	815 Dovetail Alley	1bl5-525-8f5dg-58...
17	lharesnapeg@shinystat.com	5d3e2927dd5ddddd521659cb7cbc3436	Lise Haresnape	925-304-2484	208 Bunker Hill Road	8411-787d-44f-45gd
18	nbyeh@ovh.net	99e9e9af8bc7619c747ed26f32289dde	Ninon Bye	521-582-2732	797 Eliot Drive	454eb-d554-f5d-r4...
19	cmonceyi@w3.org	c1bc8fb1d078142cffaa0e9818b6aa6a	Charles Moncey	943-403-6754	56 Mccormick Terrace	ef451-895s-45fs-5...
20	aastlej@mysql.com	81b4d83856aa1beff8eeb34bb23bd841	Adoree Astle	406-383-4369	2 Katie Plaza	215e-sv54-s5852
21	sammy_lee@gmail.com	fab459fa5a45eb2402f7cef48ca630b8	Sammy Lee	408-558-0000	254 One St, San Jose, CA 95000	3e5bf0f1-f08b-5ed...
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Code explanation:

- Frontend

```
export async function updateUser(user) {
  fetch(`http://${url}/users/update?email=${user.email}&cell=${user.cell}`+
    `&password=${user.password}&name=${user.name}&address=${user.address}`+
    `&accountID=${user.accountID}`)
    .catch((err) => {
      console.error(err);
    });
}
```

- Backend

```
app.get('/users/update', (req, res) => {
    const { email, password, name, cell, address, accountID } = req.query;
    pool.getConnection(function (err, con) {
        con.query(`update Account set email='${email}', password='${password}',
            name='${name}', cell='${cell}', address='${address}'
            where accountID=${accountID}`, (err, results) => {
            if (err) res.send(err);
            else res.send(results);
        });
        con.release();
    });

});
```

    - Updates the tuples for the given accountID with the new information.

Setup and Run Application Procedures (step by step)

1. Install npm from https://www.npmjs.com/get-npm
2. Install git from https://git-scm.com/downloads
3. Open the command prompt
4. Using git clone, clone the repository from https://github.com/CS157A-Team-30/StoreFront_Team30
5. cd into the location that repository was cloned
    a. Type “cd storefrontApplication” and hit enter
    b. Type “npm install” and hit enter
    c. Type “npm start” and hit enter
6. The application’s website should load in a browser window shortly.
7. After above steps, open a new terminal instance in the cloned repository’s directory
    a. Type “cd storefrontServer” and hit enter
    b. Type “npm install” and hit enter
    c. Create a new file named “Config.json” in the storefrontServer directory
    d. Insert the following into the file:

```
{
    "dbPort": "3306",
    "dbHost": "storefront.clcipke8dynd.us-west-1.rds.amazonaws.com",
    "username": "root",
    "password": "$aaronphuevanteam30storefront#"
}
```

    e. Type “node .” and hit enter
8. The server will now start running and accept requests from the application’s frontend

Project Conclusion

Lesson learned statement

Evan Ugarte (Team Lead)

This project taught me how to build a three tier architecture and use SQL statements in a real setting. I learned how to connect a fronted React application to a Node.js server to handle business logic, and also learned how to form SQL queries based on the requirements we needed the database to satisfy. When building the three tier architecture, my team and I also made APIs to better hide the implementation details of our database. For example, when a page needed to get the information of all items, we would call a function GetItems() instead of making

a direct query to the server, keeping the frontend code tidy and encapsulating the HTTP requests in the API functions. Overall, I gained important experience from this project that I will use in the future for other classes and in the industry.

Phu Tran (Member)

Through this project, I have a better understanding on how to apply SQL in a practical way and three-tier architecture: React.js for front-end, Node.js for back-end, and mySQL for the database. As a bonus, I also have a chance to interact with React.js and Node.js. My main work for this project is on the front-end using React.js. As a result, through many trials and errors, I now know how to intercorporate the React.js' components and rendering method with the resulting query that I get by calling a method in the back-end to display it on the web page Application. Although my job is mainly the front-end, I still have a chance to implement some functionality in the back-end using Node.js to get information from the database using the SQL query.

Aaron Warren (Member)

This project gave me a better understanding of how full stack development is done as well as how to access a database through a server. It also taught me how to make a more secure backend and some basic knowledge on database security. I also have better experience on modern ReactJS methods such as the recently introduced hooks feature which has proven to be extremely useful in the development of our website. I also obtained a better idea on frontend implementation through using functions instead of direct calls to our servers endpoints to make better looking code as well as easier to handle methods to obtain information. Overall I believe that I have gained a lot of experience in fullstack development that will greatly help my future.

Future improvement

Future improvements of this application include improving the UI of the application. To improve the UI we could switch to a different library for styling, such as MaterialUI. Another future improvement would be to make the server and frontend communicate with HTTPS instead of HTTP, improving the security and preventing sensitive data being extracted from requests. One final improvement would be to optimize the SQL queries made to our database, as we may be able to speed up the response time from the server by omitting unnecessary columns in SELECT statements, or by using INNER JOIN instead of OUTER JOIN.