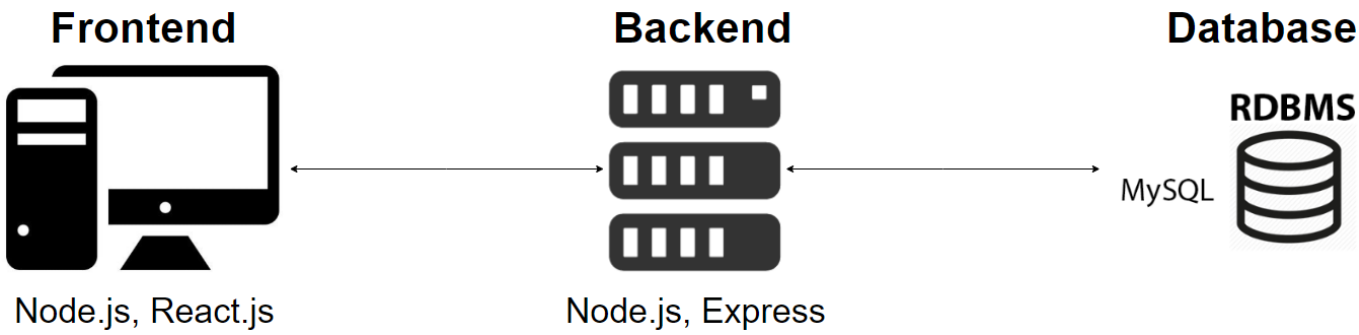
System Environment



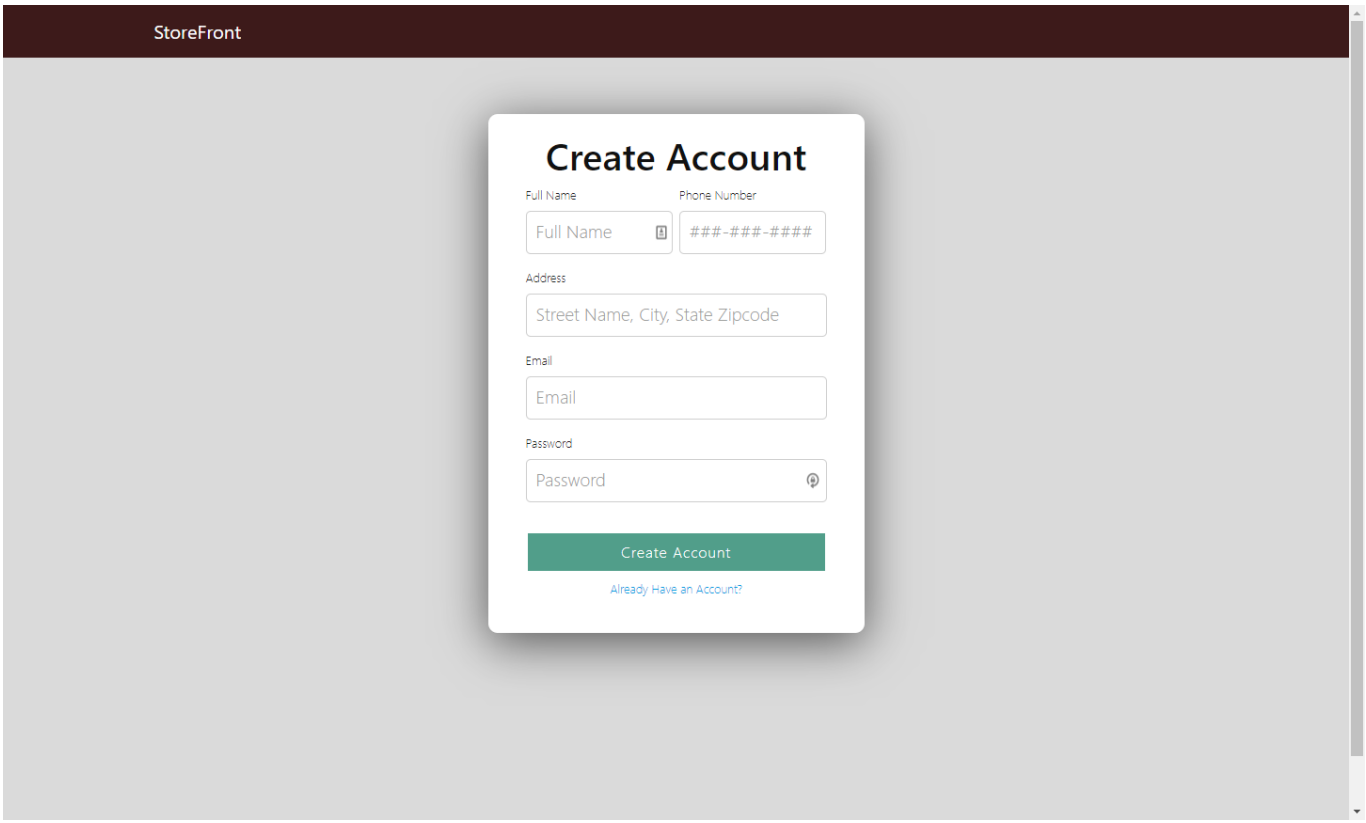
Hardware, Software used

For our 3-tier project, we made use of Amazon Web Services suite of hardware and software. Our database was run using a MySQL RDBMS and our front and backend is ran using a AWS EC2 Instance running Ubuntu Linux to process requests. The languages that are used in our project include only JavaScript and sql.

Functional Requirements

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

- Creating an Account



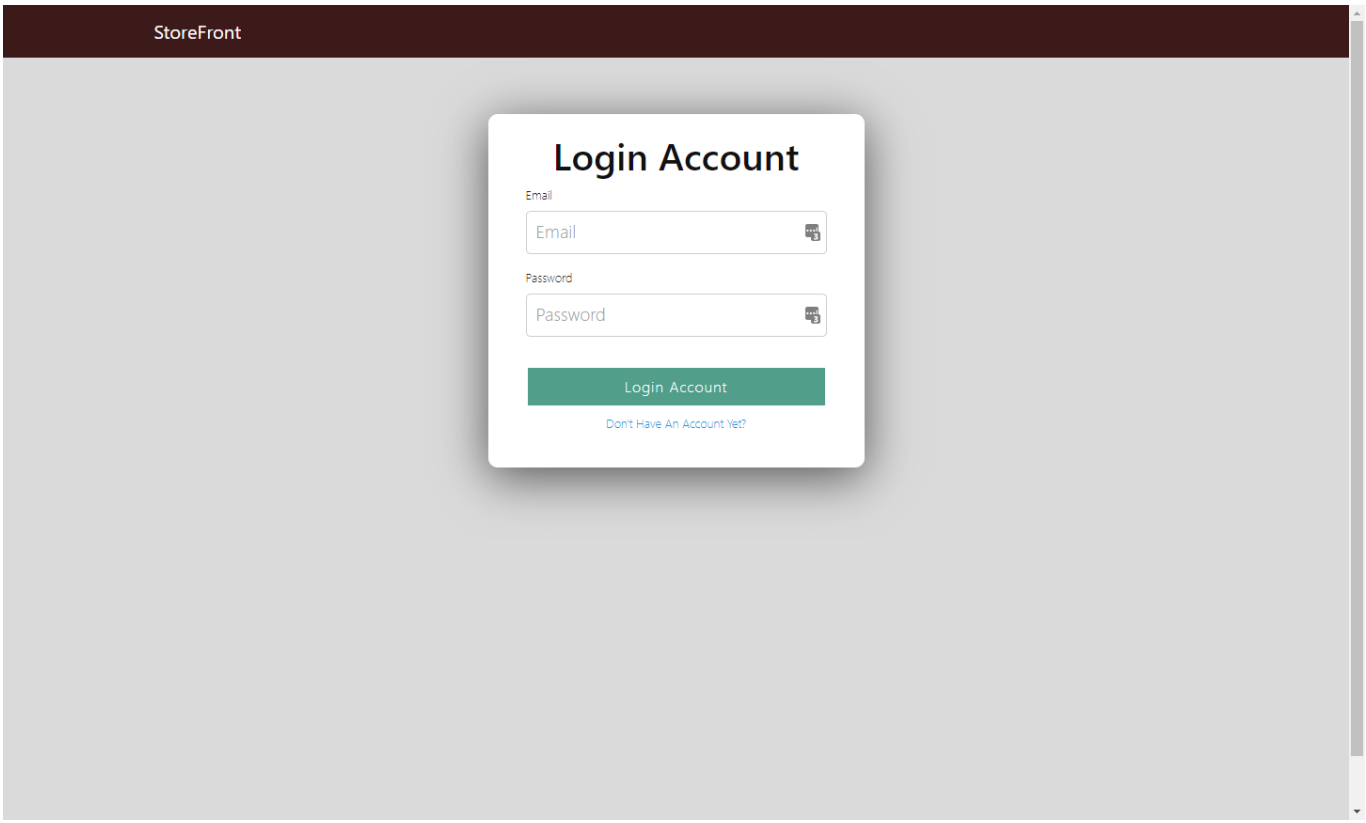
Creates an account and by taking the input information and giving it to the function `registerUser` shown below

```
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;
}
```

This function then calls a backend function that adds a user to the database shown below

```
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();
    });
});
```

- Login



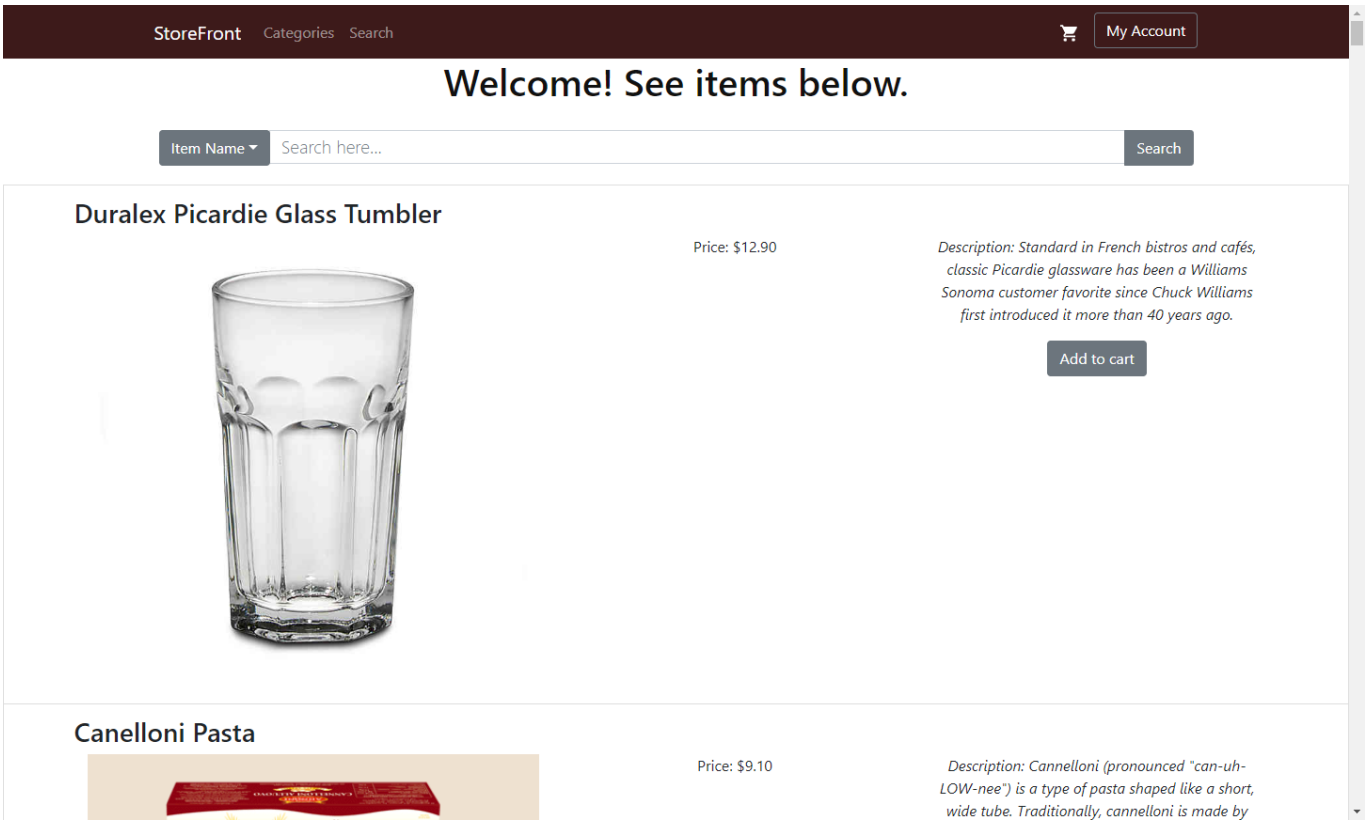
This feature just logs in a user by checking if a user exists and if it does assigns it a unique sessionID and the accountID. If it is a bad login then nothing is returned. The function called on the frontend is shown below

```
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;
}
```

Which calls on the backend function shown below

```
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();
    });
});
```

- Browse Items



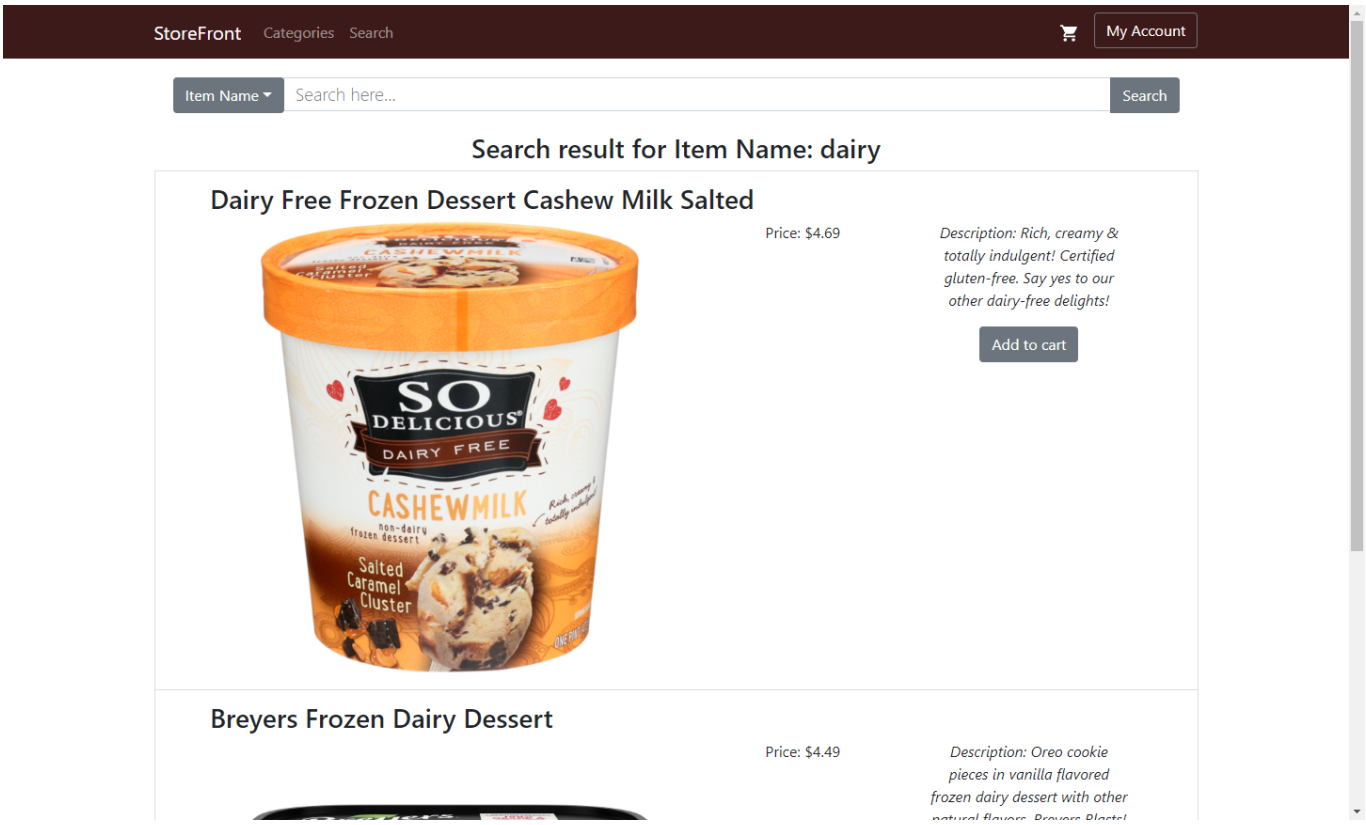
The browsing and selecting items are intertwined together with a single function call on the front and backend. What is returned is based on what is sent. If nothing is sent it will return a list of all the items, if an itemID is sent then it will return only that item, if a categoryID is sent then it will return every item in that category. The frontend functions is shown below

```javascript
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;
}
```

The backend function that is called is shown below

```javascript
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();
    });
});
```

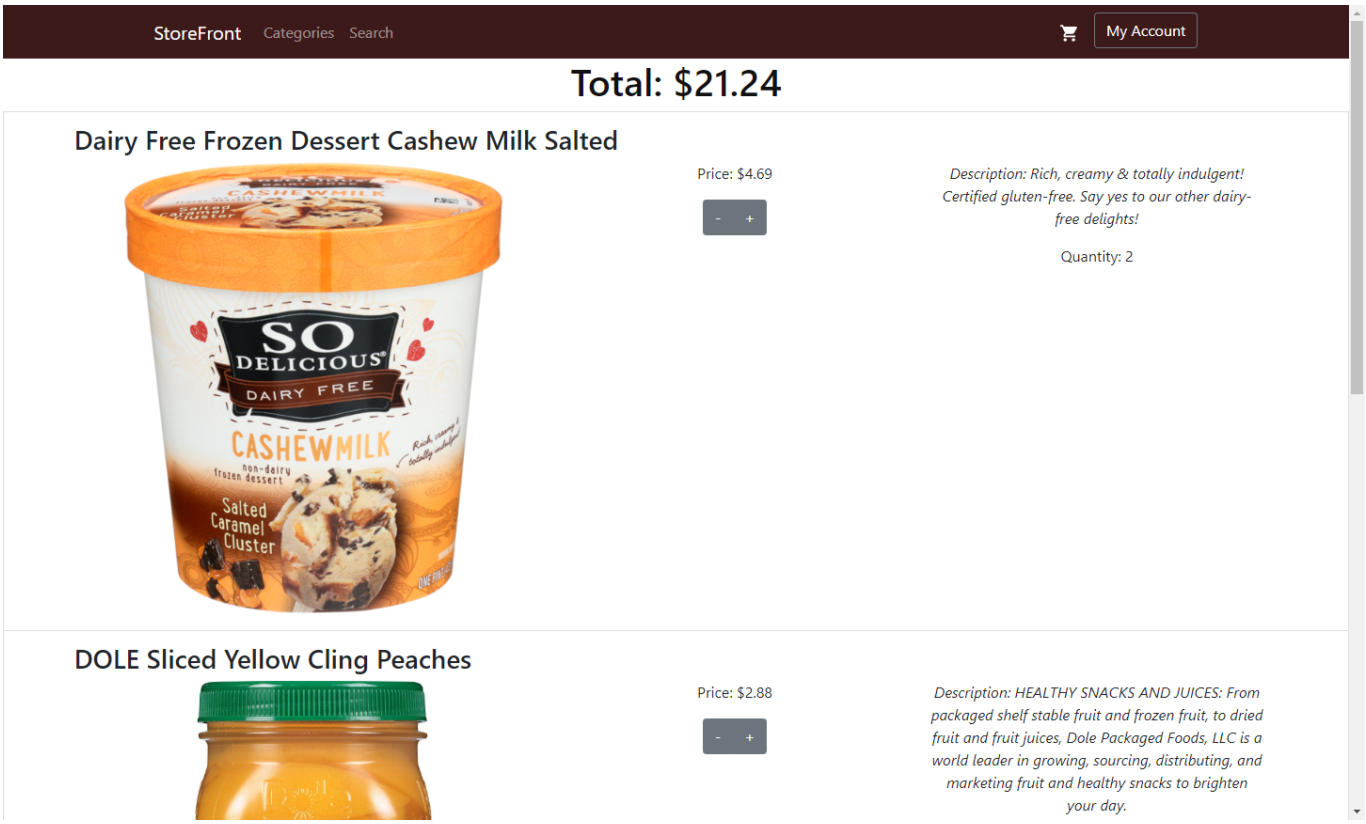
- Search Items



The search function uses the same function as the browse feature for the backend but calls a different frontend function that is shown below

```
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;
}
```

- View Cart



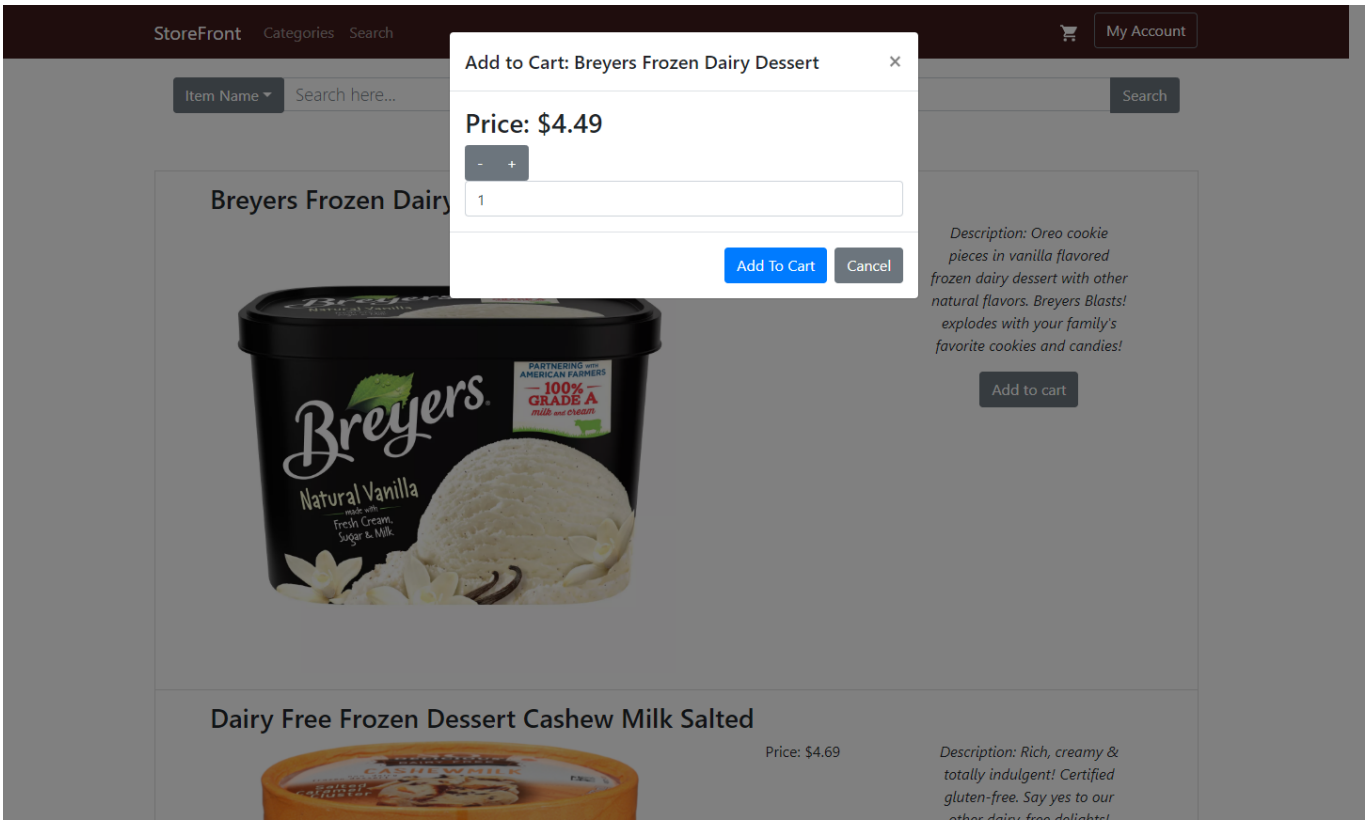
The cart shows what items a user has added to their cart while shopping. The frontend function that is called is shown below

```
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;
}
```

The backend function that is called is shown below

```
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();
    });
});
```

- Selecting and adding items to the cart

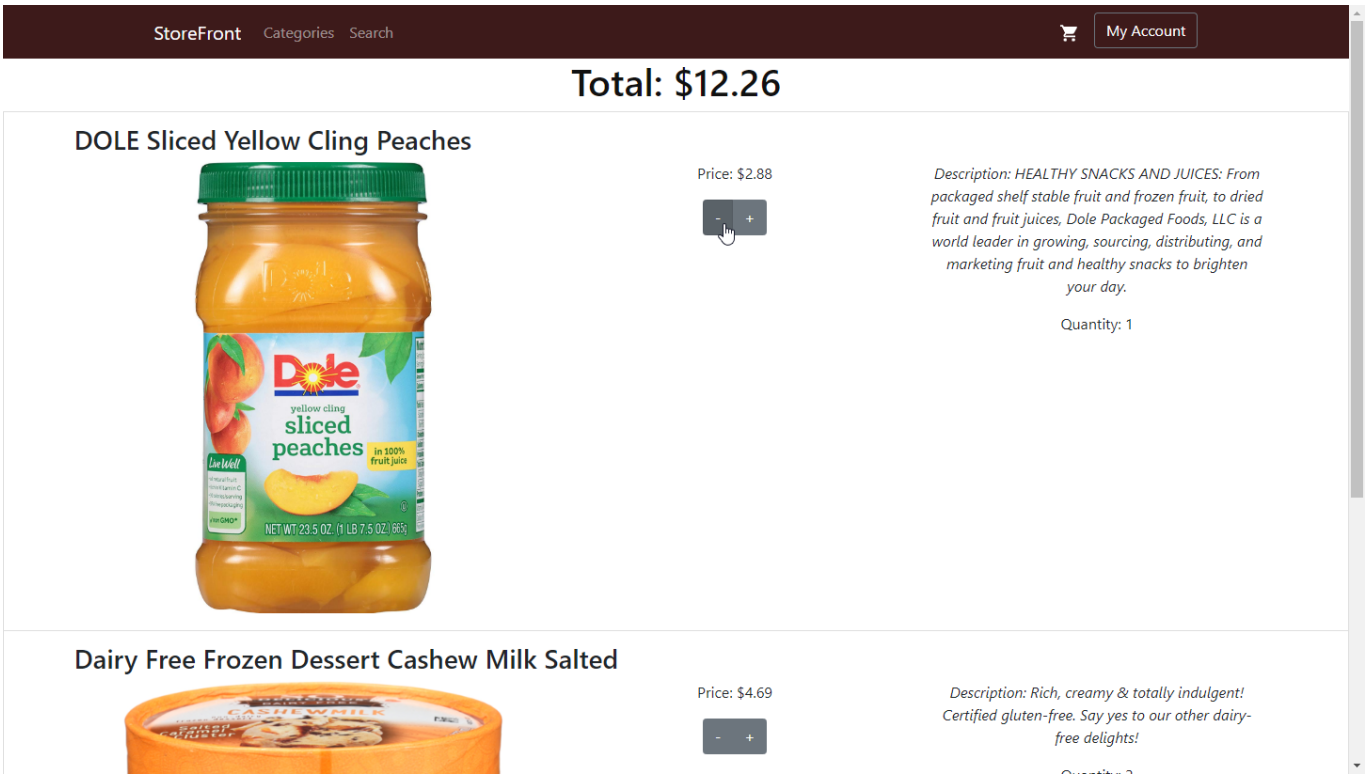


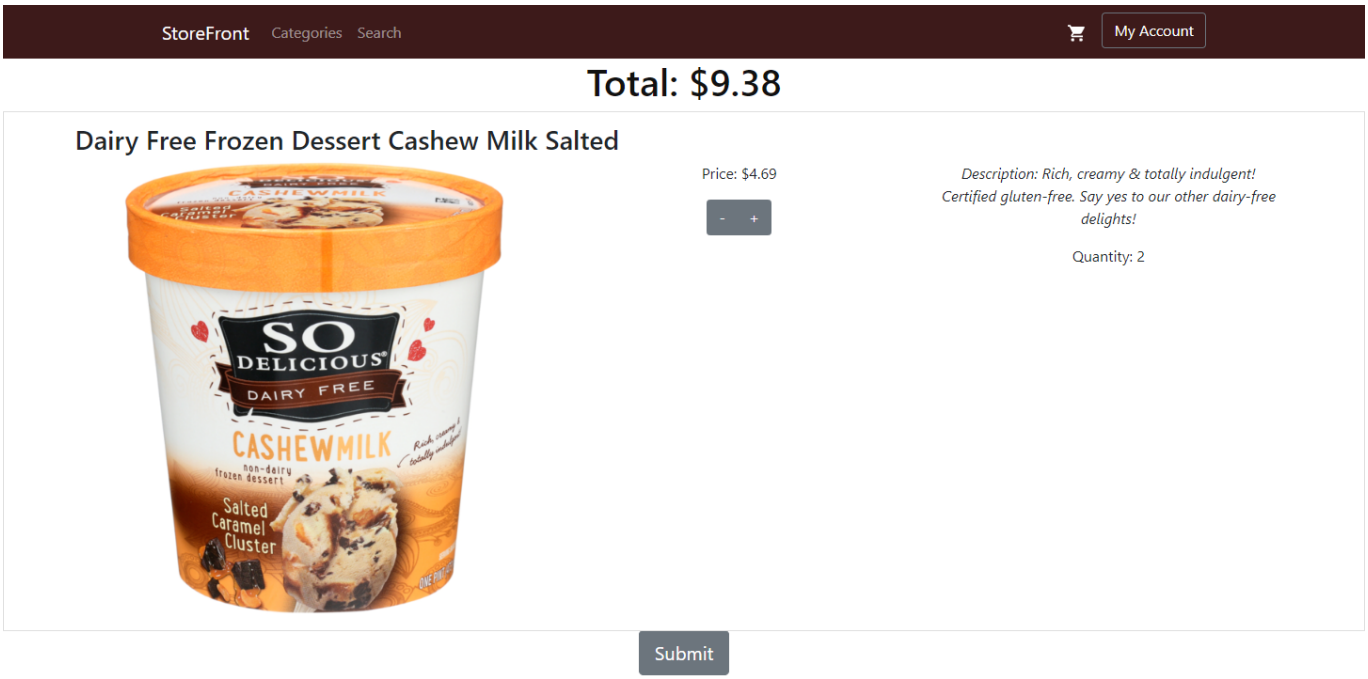
The ability to select and add items to the cart is done similarly to how we did orders. The frontend and backend code respectively is shown below

```
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;
}
```

```
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();
    });
});
```

- Deleting items in the cart



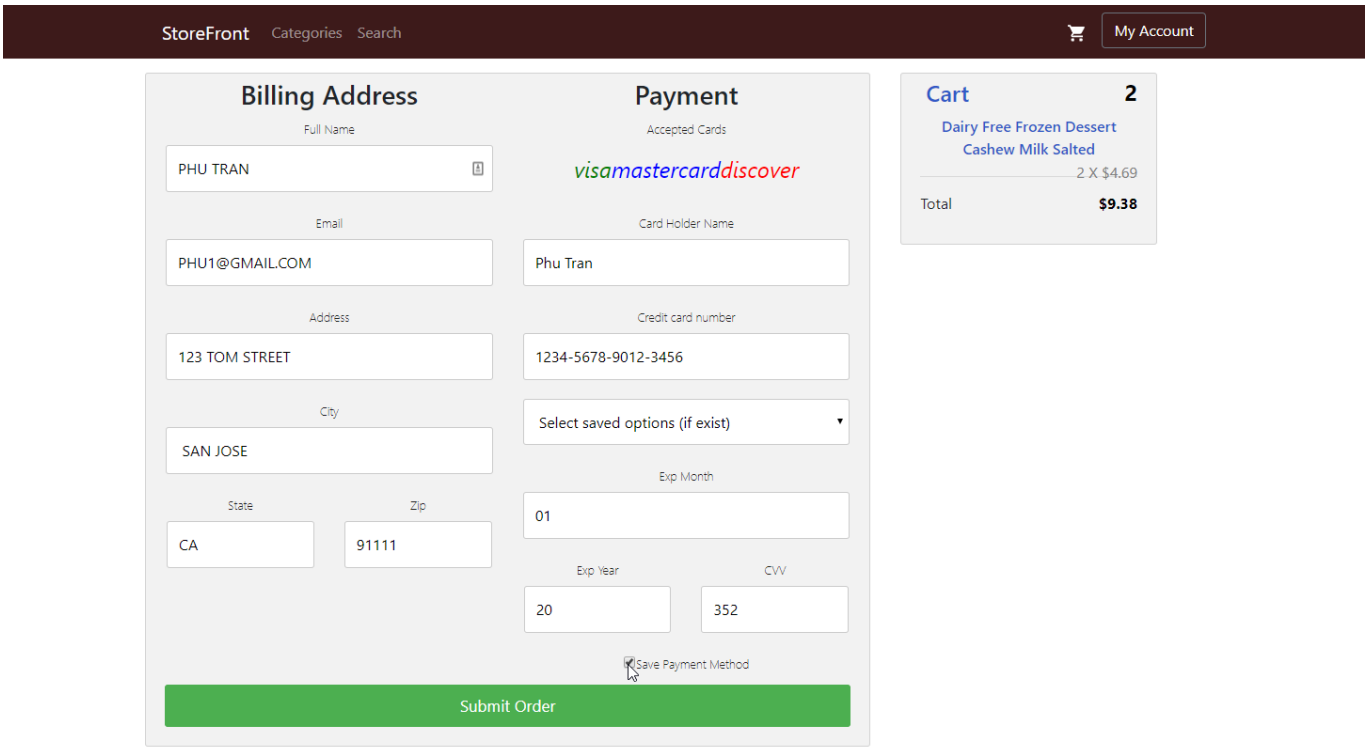


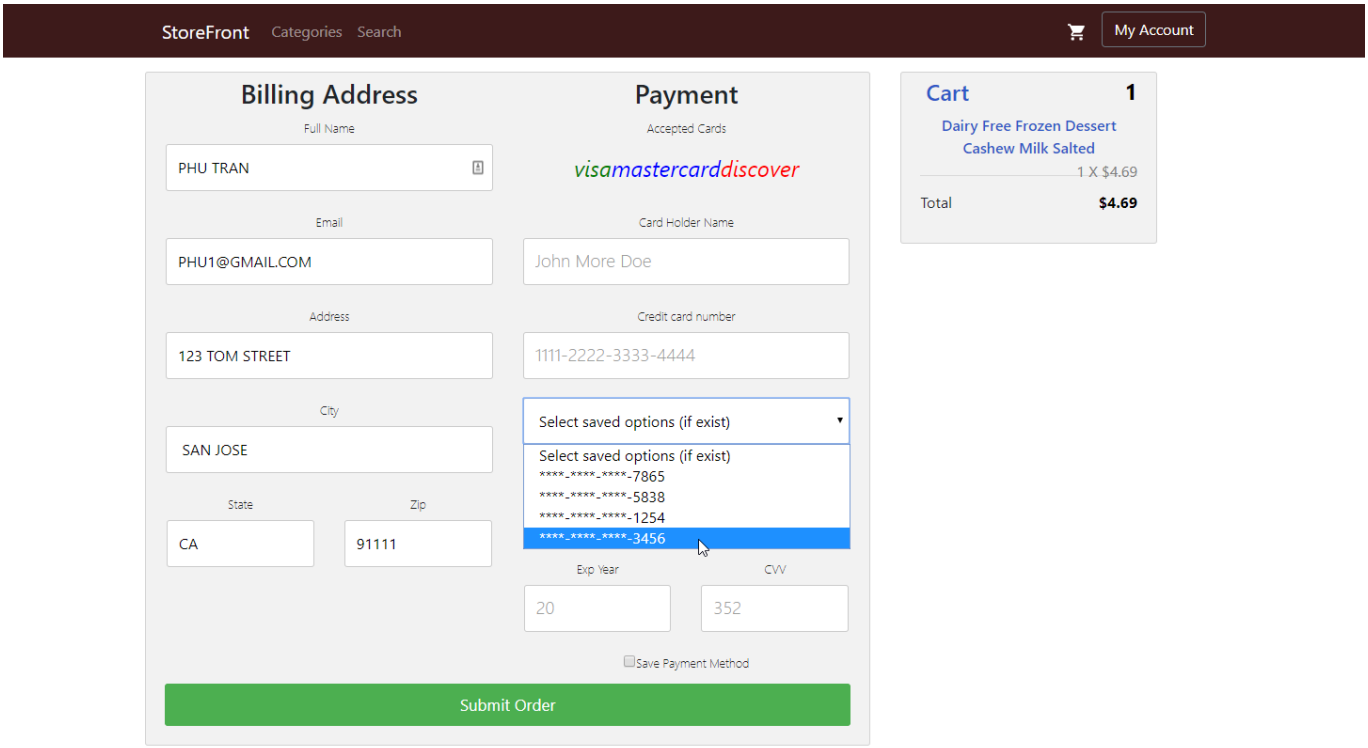
Deleting items from the cart is done very easily as it just calls a function and tells it what the itemId and quantity are to delete and sends it to the backend. Both the frontend and backend respectively are shown below

```javascript
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;
}
```

```javascript
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();
    });
});
```

- Adding new payment methods



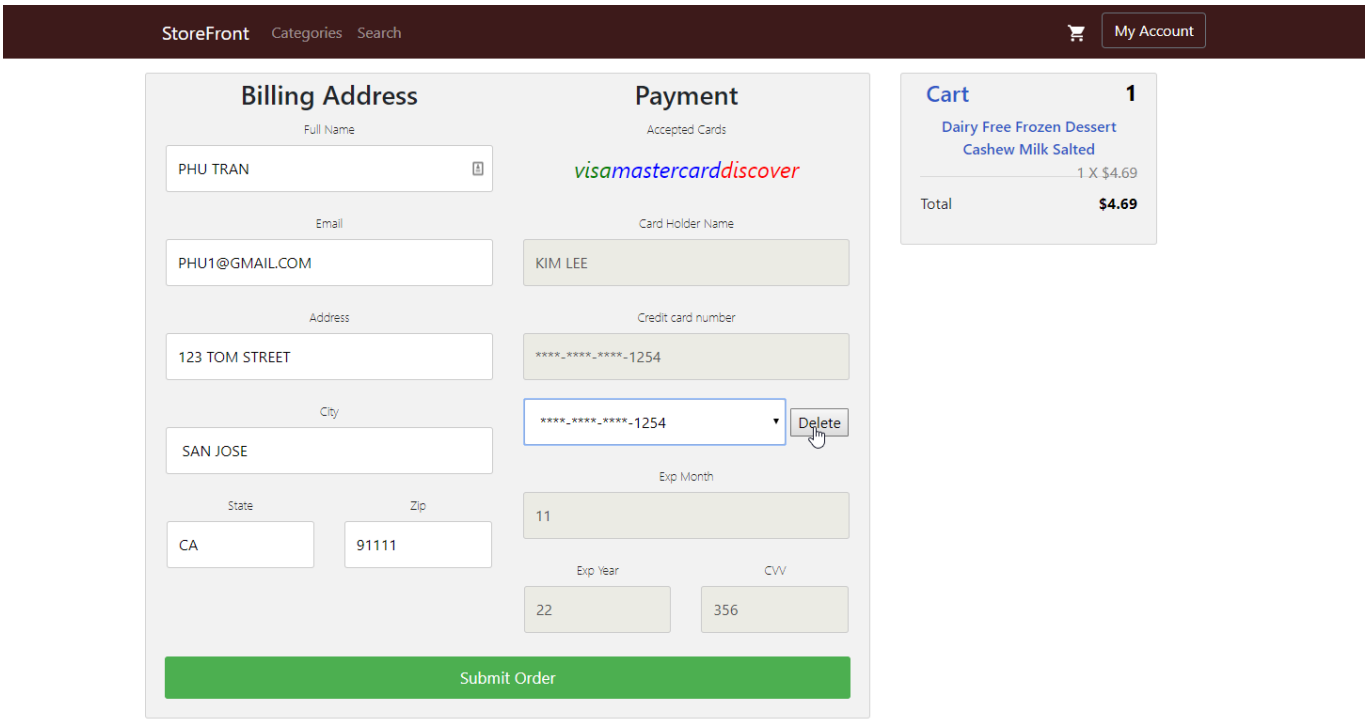


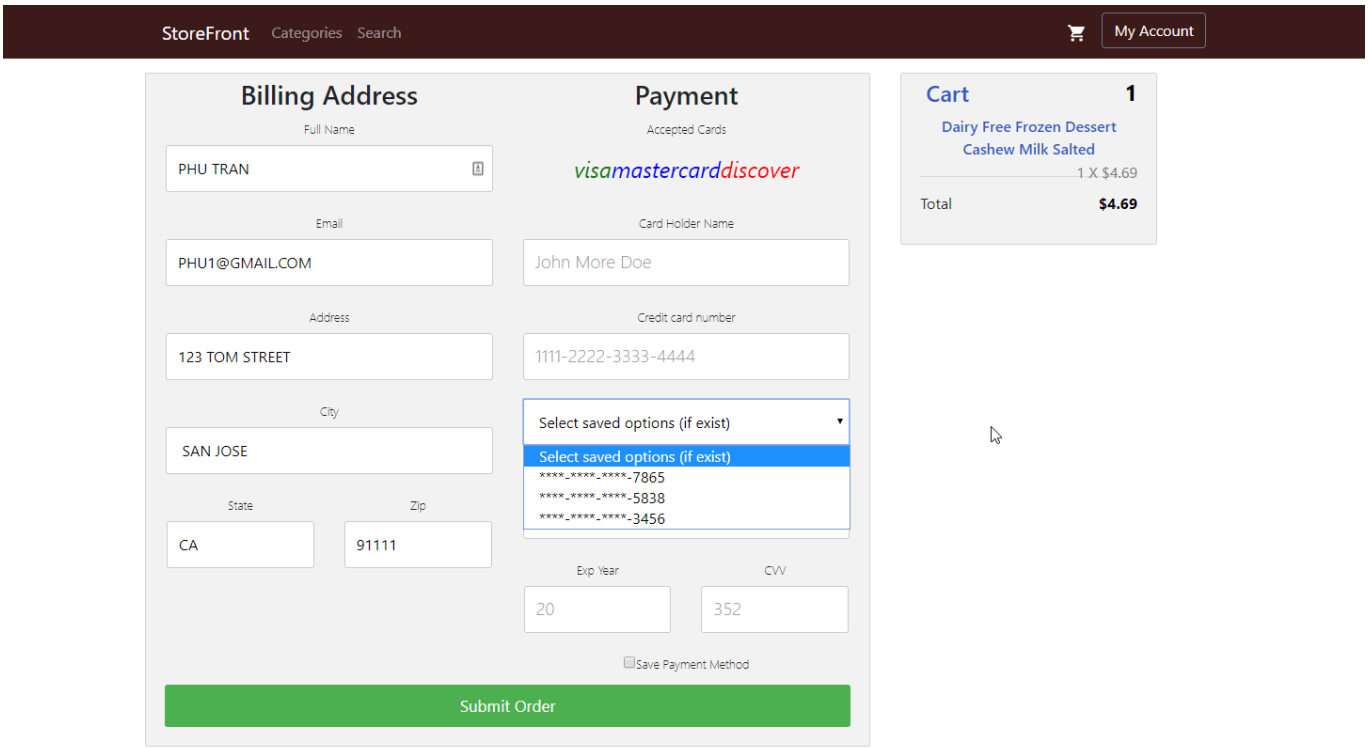
Adding a new payment method is only able to be done when a user is checking out if they wish to save a payment method. If the box is checked, then the frontend calls a function that adds the card to their account. The code for the front and backend are shown below.

```javascript
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);
    });
}
```

```javascript
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();
    });
});
```

- Deleting payment methods



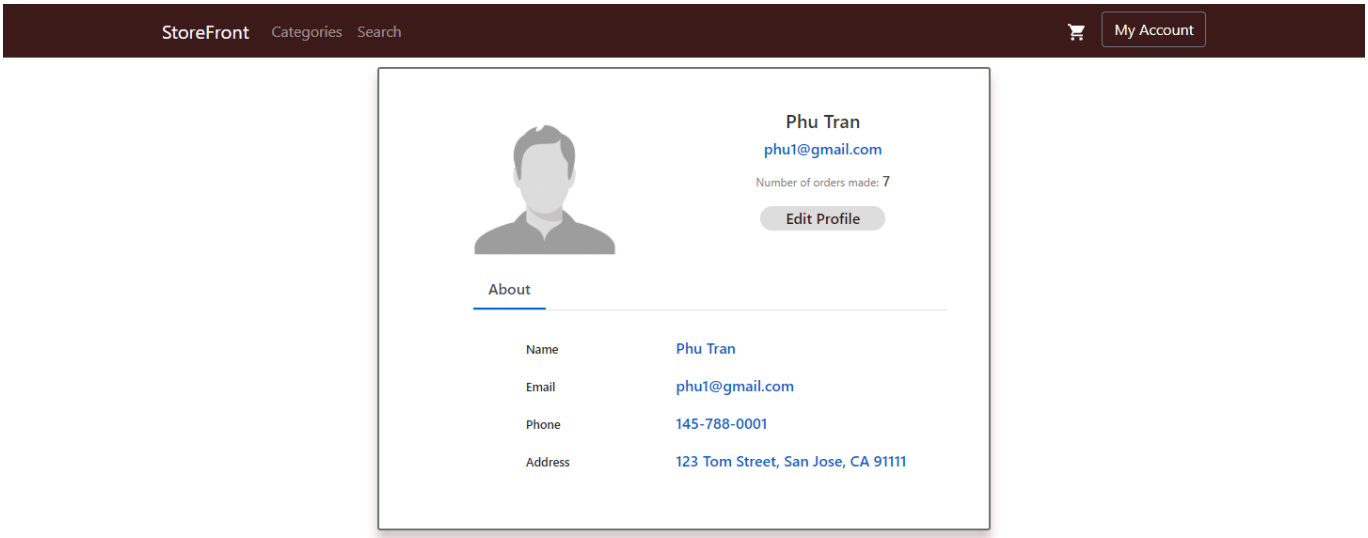


Deleting a payment method is done easily on the checkout screen as well by simply clicking the delete button. All the happens is that when the delete button is clicked, the UI will reset and a function is called to delete the information from the server. The front and backend code is shown below.

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

```
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();
    });
});
```

- View account information and email

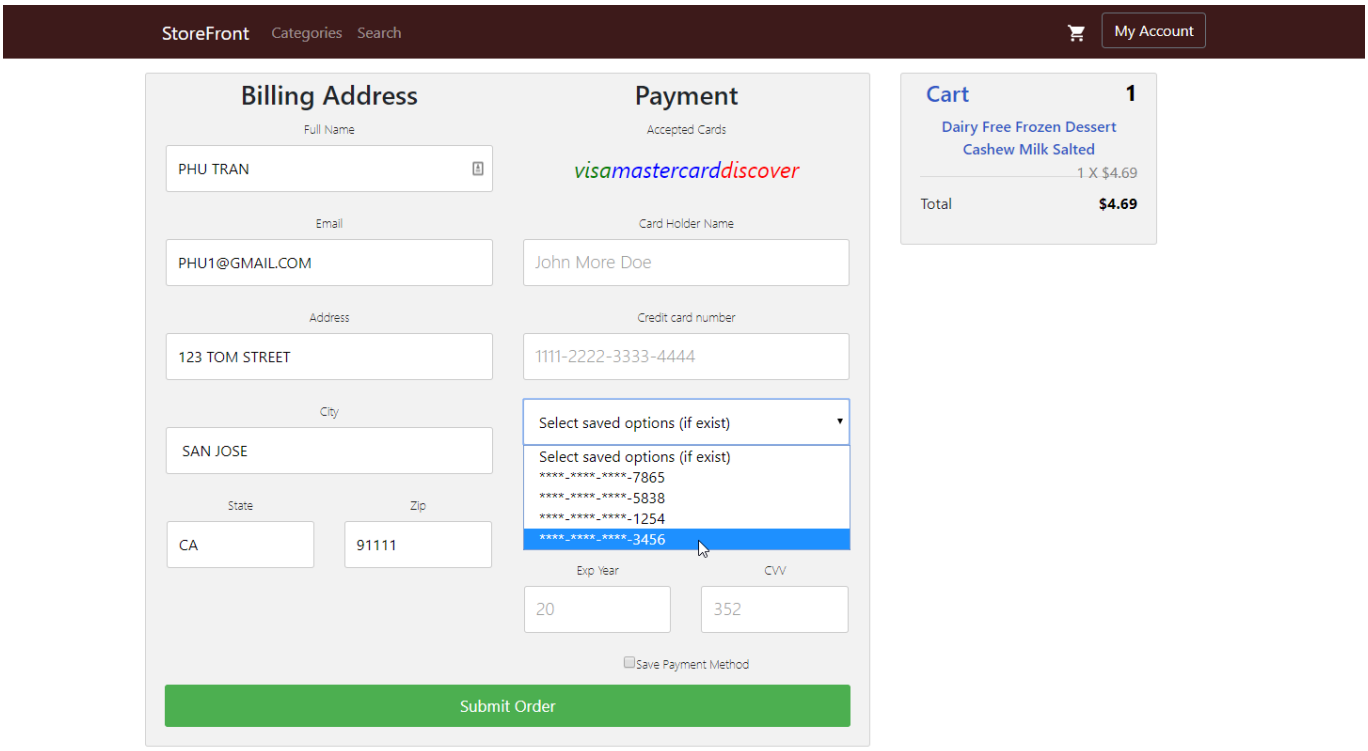


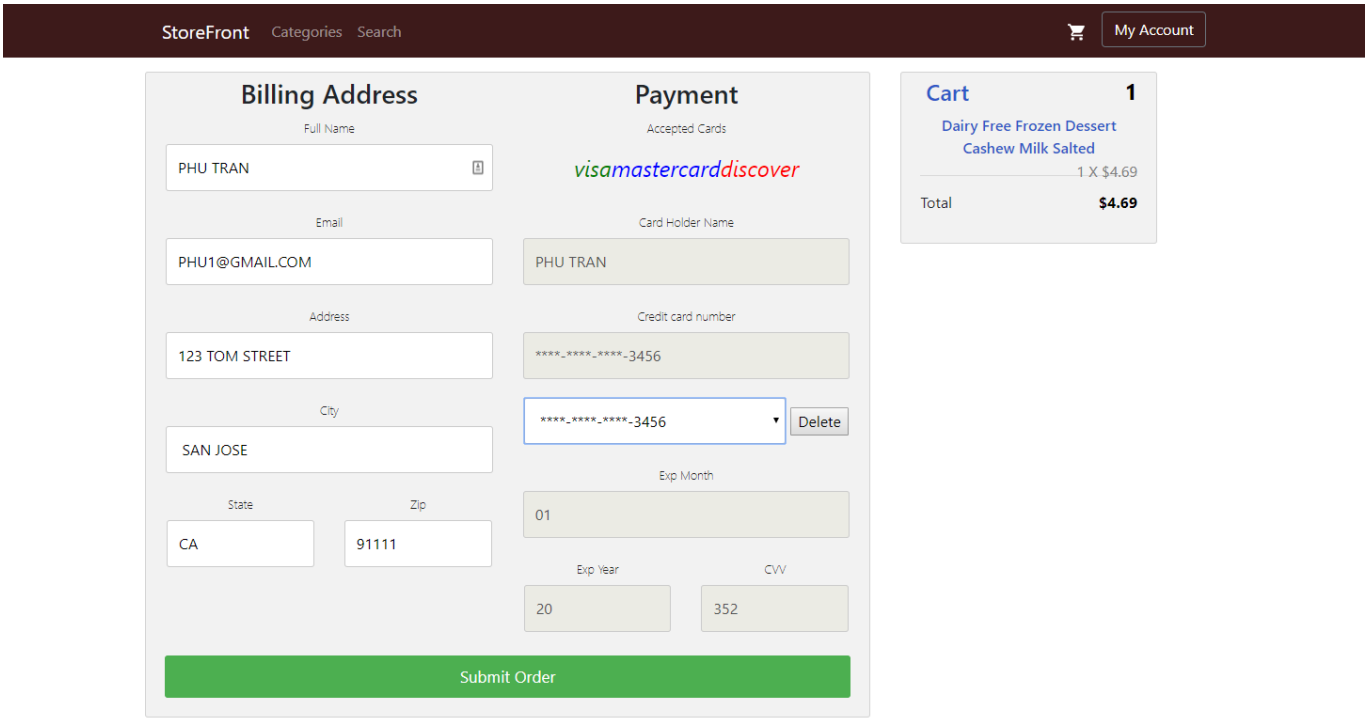
Viewing a user's account information is done by passing in the current users ID that is stored in a cookie to the backend which returns all the relevant information that is being asked for which includes the email. Front and backend code is shown below.

```
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;
}
```

```
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();
    });
});
```

- View payment methods information



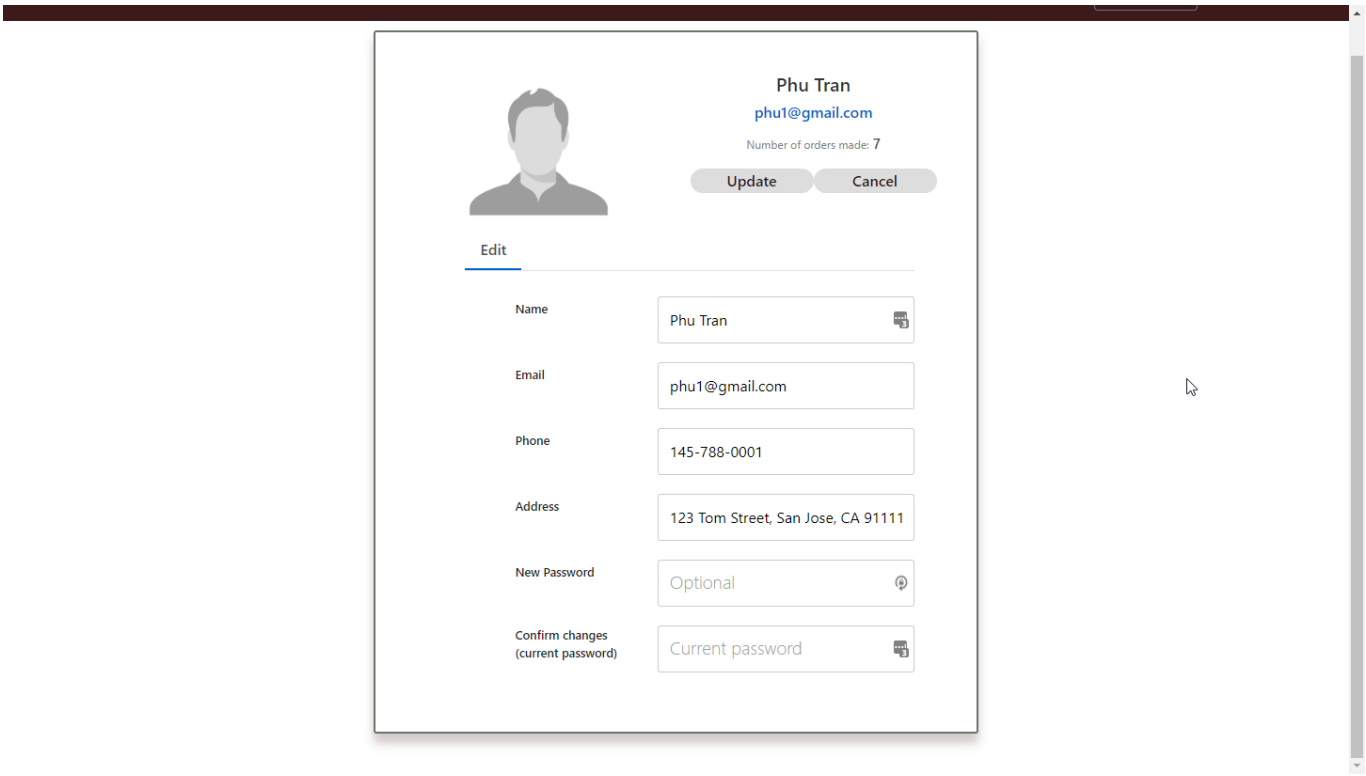


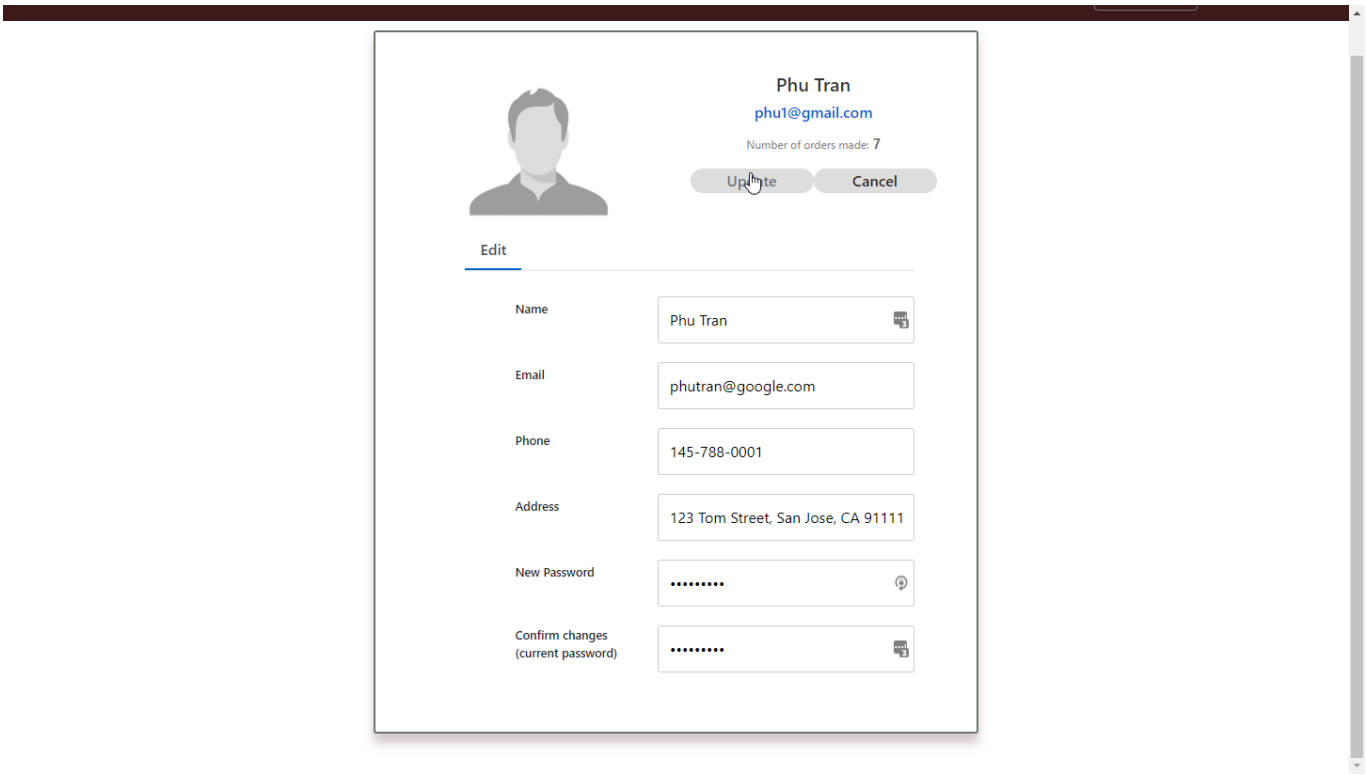
Viewing payment information can only be done on the checkout screen and is done by obtaining all payment methods that are related to the users accountID. The front and backend code is shown below.

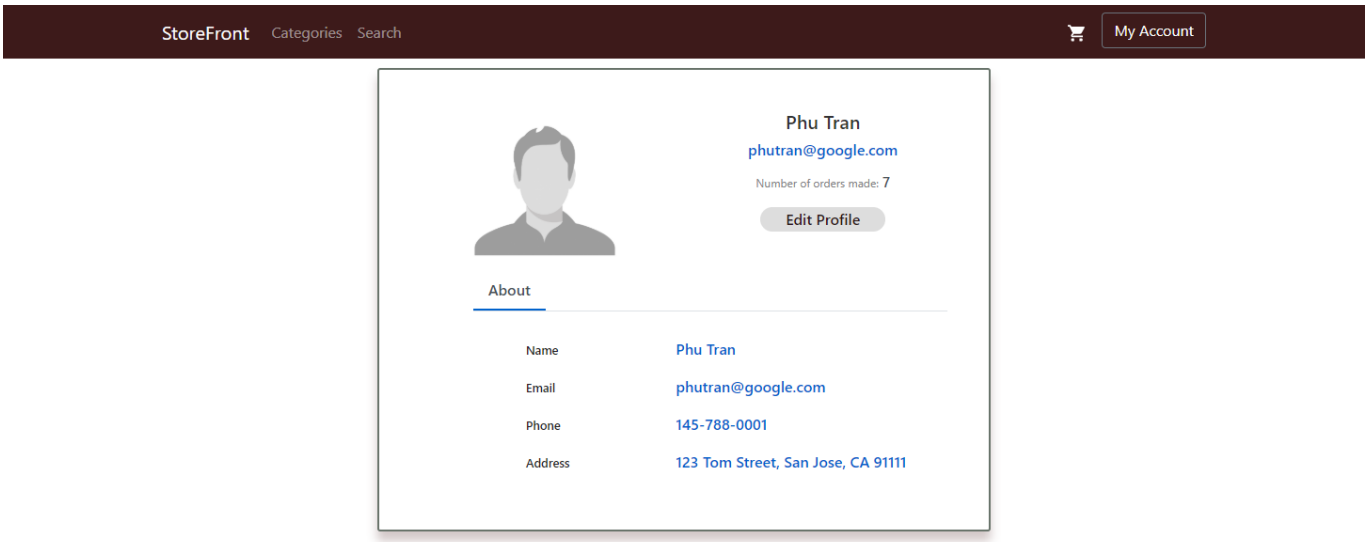
```javascript
export async function getCards(user) {
  let objects = [];
  await fetch(`http://${url}/cards?id=${user.accountID}`)
    .then((response) => response.json())
    .then((response) => {
      objects = response;
    })
    .catch((err) => {
      console.error(err);
    });
  return objects;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

```javascript
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();
    });
});
```

- Changing email and password


Phu Tran
phu1@gmail.com
Number of orders made: 7
Update
Cancel
Edit
Name
Phu Tran
Email
phu1@gmail.com
Phone
145-788-0001
Address
123 Tom Street, San Jose, CA 91111
New Password
Optional
Confirm changes (current password)
Current password


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)

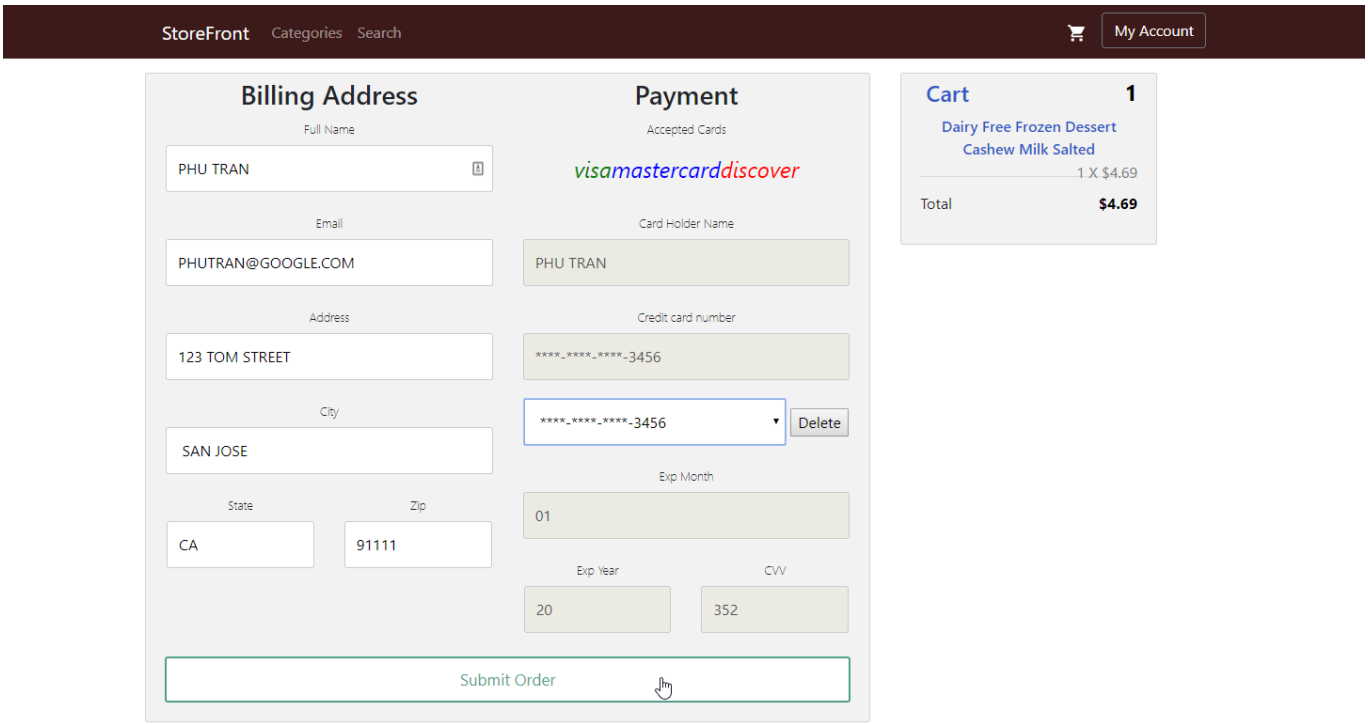


Changing the email and password can be done on the same screen. A user just has to type in a new email and password as well as confirm their new password and then hit the update button to send it to the server to confirm it. The front and backend code is shown below.

```
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);
    });
}
```


```
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);
    });
}
```


- Purchasing items

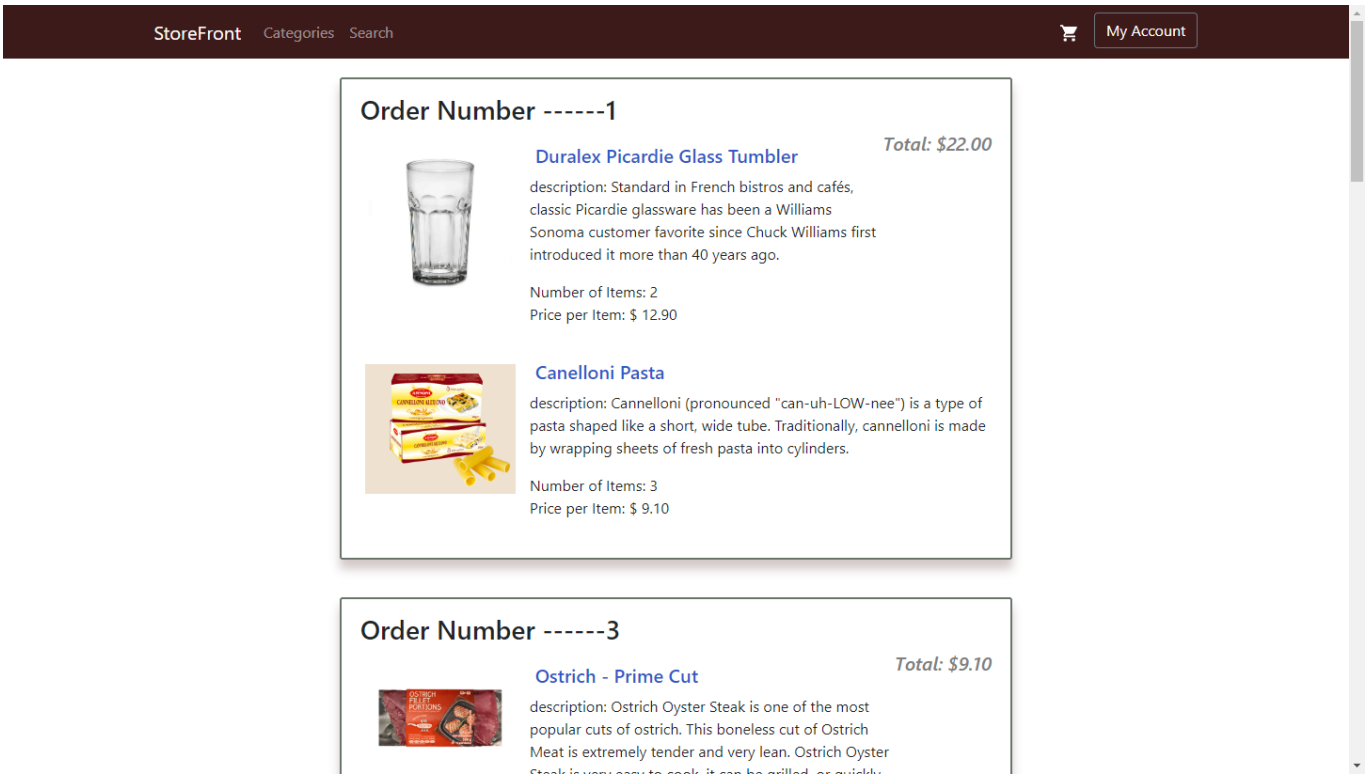


Purchasing items requires the user to go through their cart before checking out. This allows us to simply send the information stored in the cart and send it to the orders table to finalize an order. The front and backend code is shown below.

```javascript
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);
    });
}
```

```javascript
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();
    });
});
```

- View order history



Viewing order history shows whatever orders were made in the past as well as the order number, total price, and items ordered. The front and backend code is shown below.

```
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;
}
```

```
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();
    });
});
```

Implementation