CS405G Final Project Report

Matt Ruffner, Mati Turner, Matt Dunbar
github.com/ruffner/e-commerce

May 2, 2016

1 Finalized Database Design

1.1 Detailed ER diagram

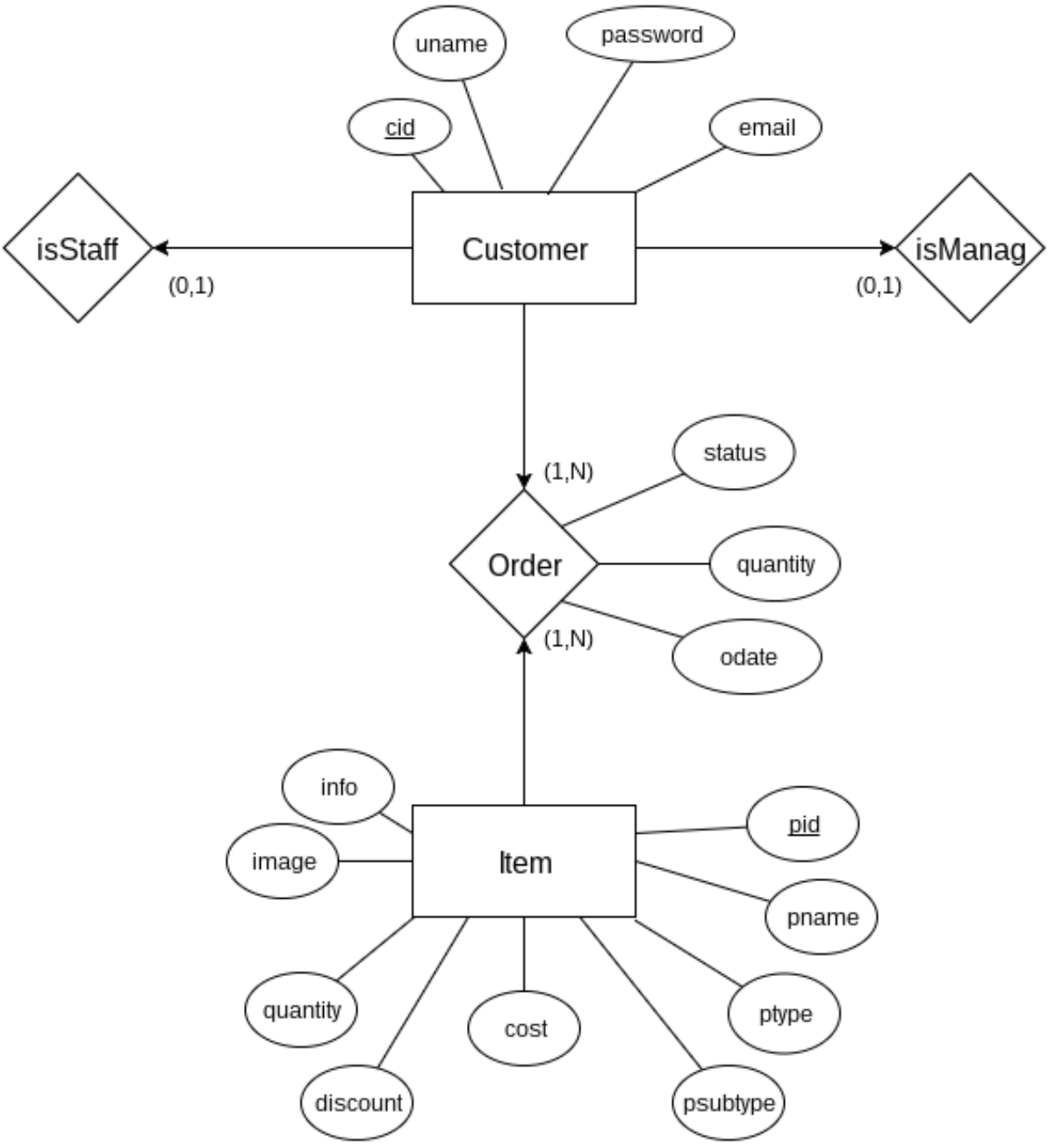

Figure 1: Final ER Diagram

1.2 Detailed database schema design

For table definitions, our code:

```
CREATE TABLE Customer (
        cid INTEGER NOT NULL AUTO_INCREMENT,
        uname VARCHAR(100) NOT NULL,
        password VARCHAR(100) NOT NULL,
        email VARCHAR(100) NOT NULL,
        PRIMARY KEY(cid),
        CHECK (LEN(password) > 5)
) ENGINE=INNODB;

CREATE TABLE Staff (
        sid INTEGER NOT NULL,
        FOREIGN KEY(sid) REFERENCES Customer(cid)
) ENGINE=INNODB;

CREATE TABLE Manager (
        mid  INTEGER NOT NULL,
        FOREIGN KEY(mid) REFERENCES Customer(cid)
) ENGINE=INNODB;

CREATE TABLE Item (
        pid INTEGER NOT NULL AUTO_INCREMENT,
        pname VARCHAR(500) NOT NULL,
        ptype VARCHAR(100) NOT NULL,
        psubtype VARCHAR(100),
        cost REAL NOT NULL,
        discount REAL NOT NULL,
        quantity INTEGER NOT NULL,
        image VARCHAR(100),
        info VARCHAR(1000),
        PRIMARY KEY(pid),
        CHECK (cost > 0),
        CHECK (discount < 1 AND discount >=0),
        CHECK (quantity >=0)
) ENGINE=INNODB;

CREATE TABLE Orders (
        cid INTEGER NOT NULL,
        pid INTEGER NOT NULL,
        status VARCHAR(100) NOT NULL,
        quantity INTEGER NOT NULL,
        odate DATETIME,
        FOREIGN KEY(cid) REFERENCES Customer(cid),
        FOREIGN KEY(pid) REFERENCES Item(pid),
        CHECK (cid IN (SELECT cid FROM Customer)),
        CHECK (pid IN (SELECT pid FROM Item)),
        CHECK (status="cart"),
        CHECK (quantity > 0)
) ENGINE=INNODB;
```

From the ER Diagram, its a pretty simple step to turn it into tables. The manager and staff tables simply hold the mid and sid as references to the customer entity they stem from. The order table contains the part and customer IDs that make up the relationship. The customer and item entities translate directly into tables. The status trigger in the orders table makes sure that upon creation, the item is listed in a cart because that is how the program flow dictates any entry should originate.

1.3 Functional Dependencies

Manager and staff tables are omitted from the relational dependency analysis below, for theyre really just customer tables.

1.3.1 Customer

Let $A = cid, B = uname, C = password, D = email$. Then $A \to B, A \to C, A \to D$.

From the key, all values can be determined, there are no dependencies on any non-key attribute. Customer table is 3NF.

1.3.2 Order

Let $A = cid, B = pid, C = status, D = quantity$. Then $AB \to C, AB \to D, AB \to E$

From the candidate key obtained by combining the cid and pid of the order, we can find the other members of the table. There arent any dependencies within the row among the data members, thus each can be determined by the combination of the keys. The Orders table is 3NF.

1.3.3 Item

Let $A = pid, B = pname, C = ptype, D = psubtype, E = cost, F = discount, G = image, H = info$. Then $A \to B, A \to C, AC \to D, A \to E, A \to F, A \to G, A \to H$.

Every model can be determined by the pid or the combination of the pid and type (in the case of subtype). Thus there arent any non-trivial dependencies on anything other than the key or a superkey. As such, the Item table is 3NF.

2 Description of programs

2.1 Program flow

This web application is designed on an asynchronous model which preforms all data transfer via AJAX requests, eliminating the need for page reloads. The site is constructed using Google's material design web components, the PolymerElements.[**?**]

When a customer visits the site, a PHP session variable is populated which contains the database connection which may be queried upon subsequent requests.

Various PHP pages were purpose written and separated by functionality. Templating and databinding, offered by Polymer, allow control over DOM visibility, making it easy to show 'admin' views only if the corresponding variable is set in the user structure/object. This may see like a vulnerability since the JavaScript object can be modified after pageload. This is not the case however, since all PHP pages which preform actions requested by admin views (order submitting, item updating/shipping, etc) first verify that the user requesting the actions does indeed have permission to.

2.2 Data structures

The PHP array seen in Fig. **??** is stored in a `$_SESSION` variable, it holds user information and is passed back with every AJAX request to the front end. By specifying the header of the PHP page's output to be JSON,

with `Content-Type: application/json`, arrays like the one seen in Fig. **??** are converted to JavaScript objects. To send the array below back to the front end, the PHP would be `echo json_encode($BLANK_USER)`.

The Iron Element `iron-ajax` is used to handle asynchronous requests. Dot notation is then used on the font end to access the information. An `iron-ajax` might be implemented as seen in Fig. **??**. The `on-response` tag specifies a callback for the data to be passed to. To extract the users email address from the received data, assuming the signature `handleResponse(e)`, would then be `e.detail.response.email`

```
$BLANK_USER = Array(
        "uname" => "",
        "email" => "",
        "isStaff" => False,
        "isManager" => False
);
```

Figure 2: PHP structure to hold user information.

```
<iron-ajax
    url="../src/products.php"
    handle-as="json"
    on-response="handleResponse">
</iron-ajax>
```

Figure 3: Example

2.3 Algorithms

One interesting portion of code is the JavaScript which processes the orders from the SQL table to be shown in the order cards for the user. An image of the order card can be seen in Fig. **??** In the database, there is one row for each item in an order, with unique dates grouping items in the same order (*odate* is NULL if the item is in the users cart).

```
computeOrders: function(o) {
  var g = []; // to hold orders grouped by order date

  if(o && o.length ){ // if we have orders to show
    var time = '';
    var total = 0;

    // go through each order
    for( var i=0; i<o.length; i++ ){

      o[i].price = o[i].price * o[i].quantity;  // adjust price

      // if this is a new odate group
      if( time != o[i].odate ){
        // new order but not first,  assign total thusfar
        if(total)
          g[g.length-1].total = total;
        total = o[i].price;

        // push a new model, representing one order card each item in the items
```

```
        // list gets iterated over by the inner template
        g.push({
          odate: o[i].odate,
          cid: o[i].cid,
          items: [ o[i] ],
          status: o[i].status,
          statusMessage: "",
          pending: (o[i].status == "pending"),
          total: total,
          uname: o[i].uname,
          email: o[i].email
        });

        time = o[i].odate
      }
      // an item from the same order
      else {
        g[g.length-1].items.push( o[i] );  // add it to the most recent items array

        total += o[i].price; // add its price to the total

        if( i == o.length-1 ) // assign the last total since loop wont go again
          g[g.length-1].total = total;
      }
    }
    return g;
  } else {
    console.log("no orders to group");
  }
}
```

3 Program functions

3.1 Sample input and output screens for each function

3.1.1 Login Screen

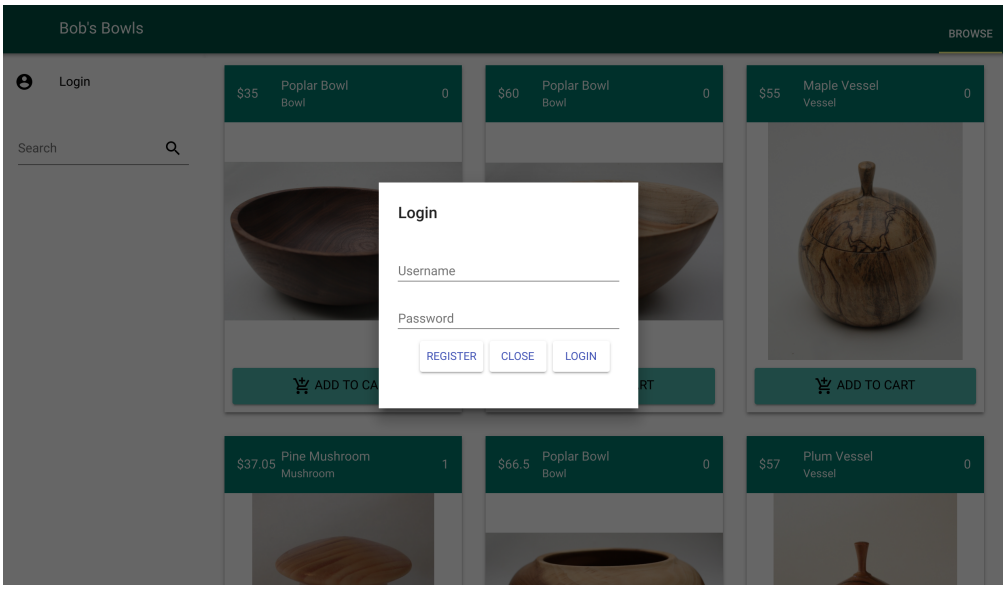

Figure 4: The Login Screen

3.1.2 After User Login

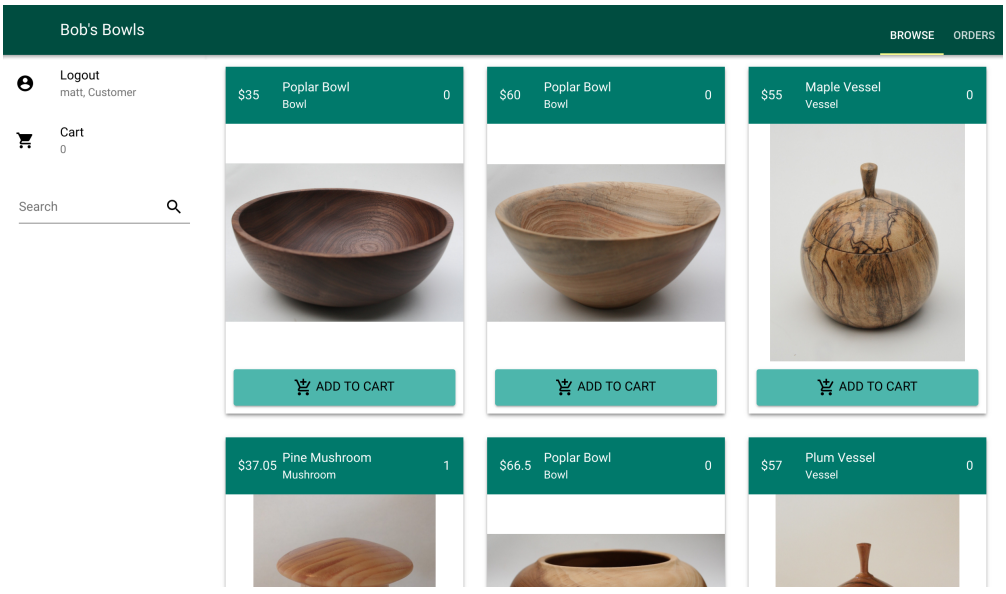

Figure 5: What a lowest access level user sees after logging in.

3.1.3 After Manager Login

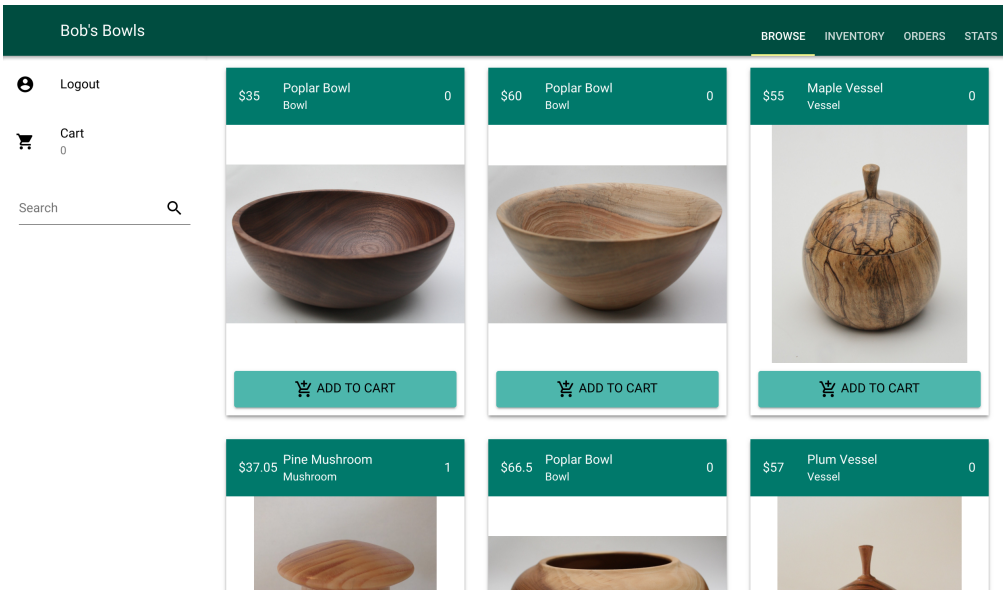

Figure 6: What a manager sees after logging in.

3.1.4 Search

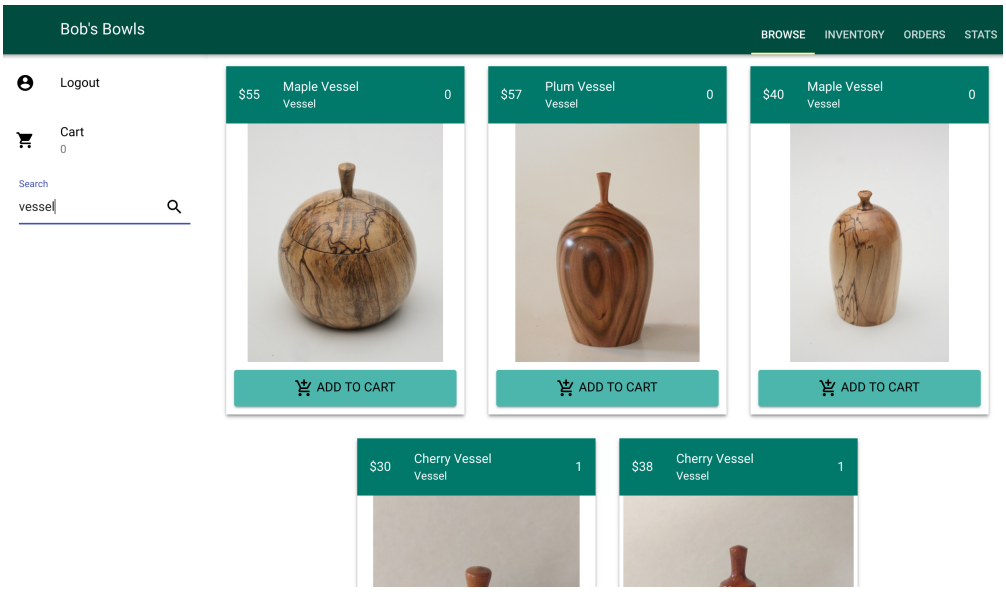

Figure 7: What filtering by type looks like.

3.1.5 Cart

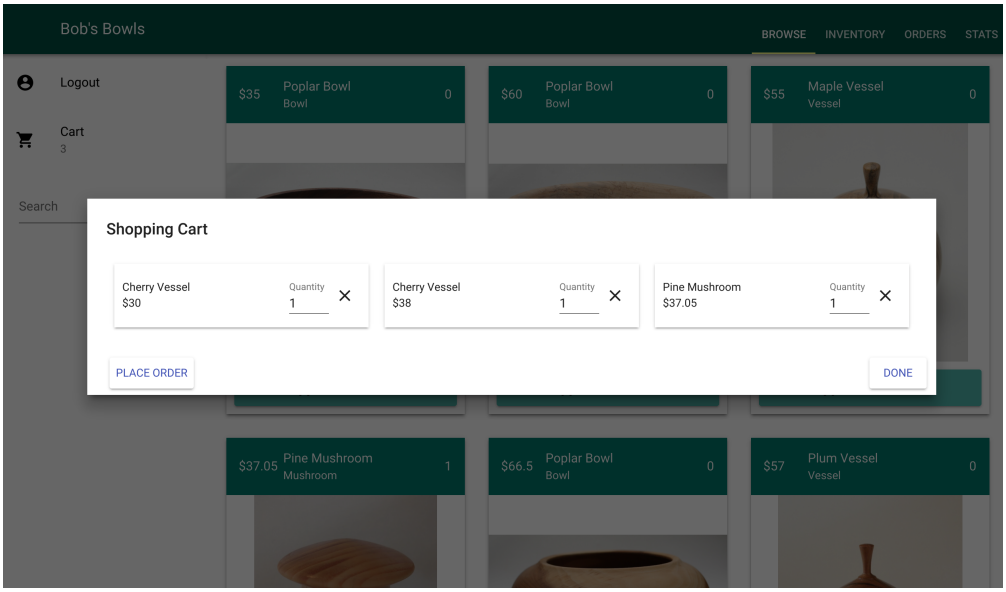

Figure 8: How items appear when in the cart.

3.1.6 Users Orders Page

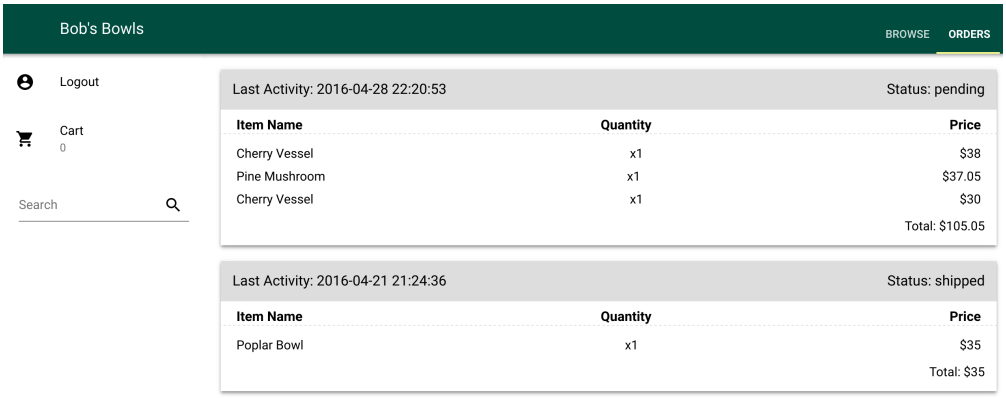

Figure 9: Viewing past orders as an ordinary user.

3.1.7 Staff Orders Page

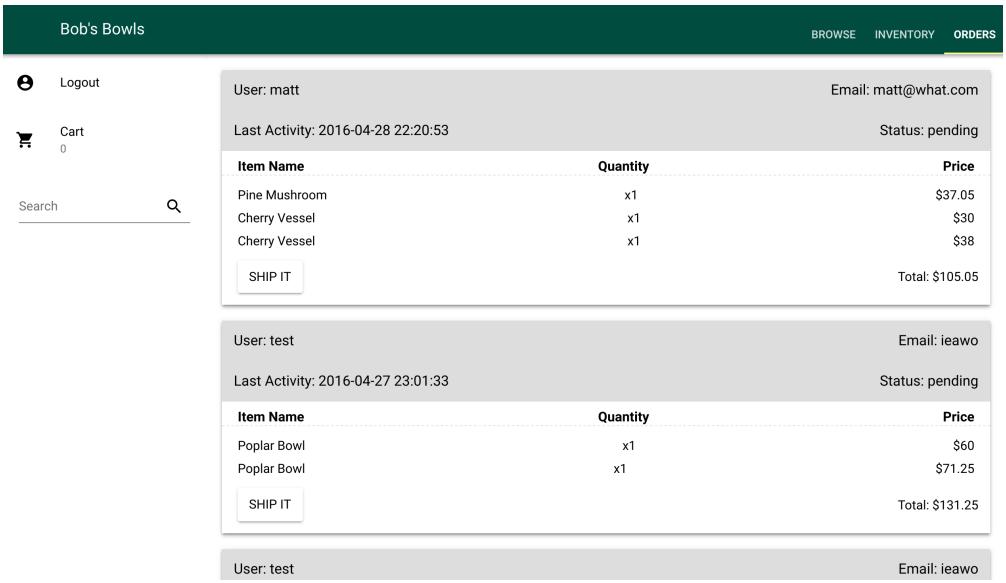

Figure 10: Viewing orders as a staff member. Note the 'SHIP IT' button.

3.1.8 Update Inventory

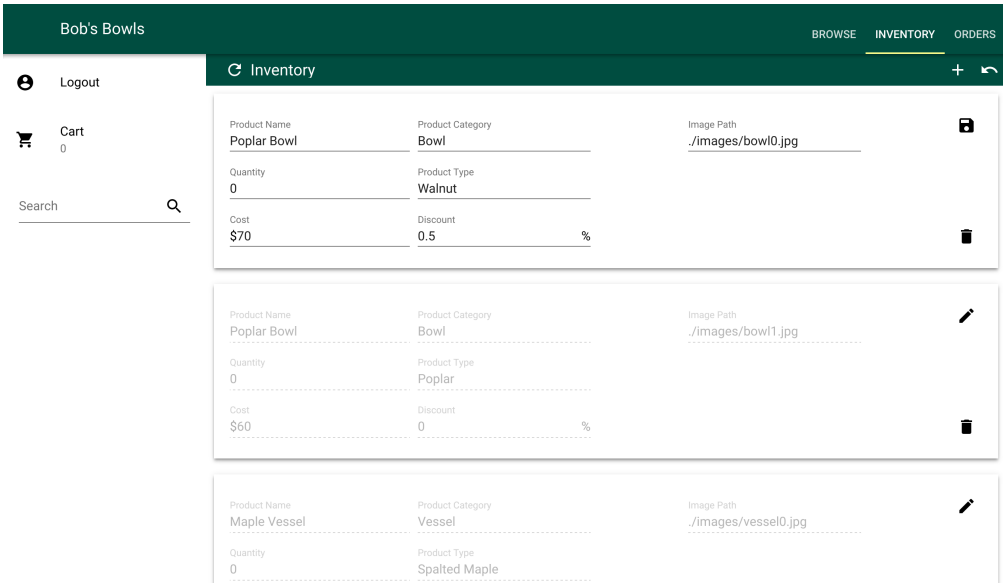

Figure 11: Editing an inventory item. The item being edited is darkened.

3.1.9 Statistics Page

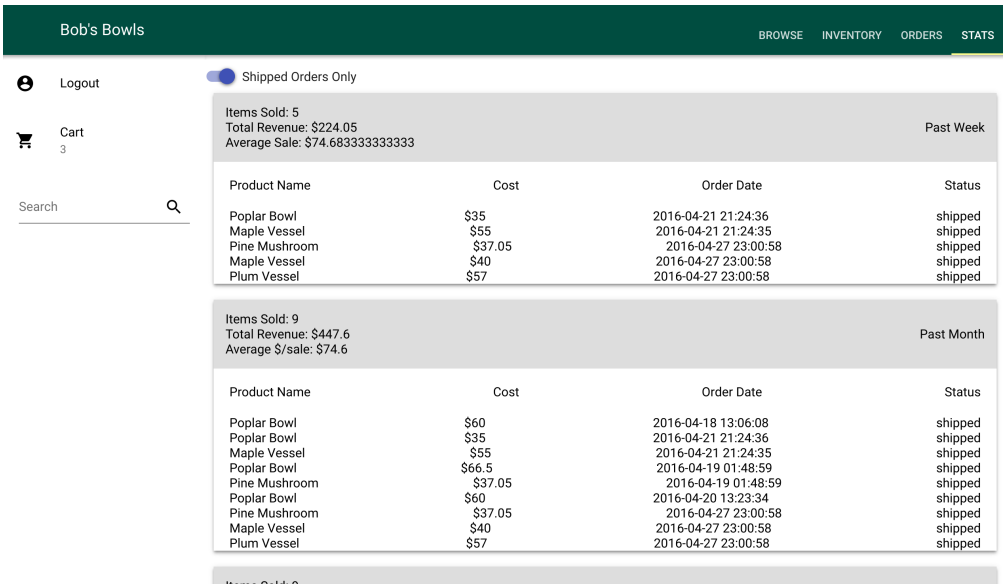

Figure 12: The statistics view for managers only.

4 Testing

4.1 Explain what you have tested to make sure your software works correctly

In testing our database application, we took the assumption that there could be no SQL injection with our front end. Thus, we only have to handle faulty input, not malicious input. In testing normal interaction with the database we simulated a moderate amount of commerce traffic in using the application as a customer, staff, and manager user.

We tested the effectiveness of our trigger statements by trying faulty input (entering negative quantities and discounts greater than 100% or less than 0%). Our triggers caught the faulty input and handled it appropriate by setting the value to something valid. In the case of discount, it rejects the input and sets back to the prior discount. On the case of negative quantity, the quantity is set to zero instead. Weve used it on multiple to make sure that the updates to inventory and price occur with a refresh or page change in real time.

4.2 Describe your project experience

In our experience, the project was mostly a learning experience for SQL and database design. In contrast it was most a chance to show off some depth of knowledge of website design. The pre-existing knowledge of programming interactivity and websites proved invaluable for showing off a good database foundation. The work itself was long but easy to spread out. Communicating the needs of the different parts of our project was key as interfacing between the two took longer than expected. The project truly showcased how important it is to balance all the moving parts of a web application.

References

[1] Polymer Element Catalog
`https://www.polymer-project.org/1.0/`