CPSC 471 Database Management Systems

Final Report

Group 19

Abstract

The advent of online retail shopping has introduced many platforms on the internet, such as Amazon, Alibaba, and eBay. This has led to creation of inventory systems and retail system that requires complex database structures and complex queries to facilitate the massive platforms. In our project, we tried to capture some of the basics of the complexity that enables buyers and sellers to interact while having employees manage the system. We used MySQL DBMS hosted on Amazon Web Services platform along with PHP frontend to accomplish this task. Our platform allows users to register as buyers and allow registered users to upgrade to sellers. It allows users to search, and save items listed by sellers in their shopping cart as well as place orders from different sellers. Sellers are given additional ability to manage their inventory, update product listing, and ship orders. The platform also allows the moderators or employees to add new departments, and upgrade users. This was accomplished by combination of simple CRUD (create, read, update, delete) SQL statements, in addition some complex statements such as INNERJOIN and LIKE query parameters were also used to retrieve and manipulate data in our database.

Introduction

The goal of our database is to create an online environment where users can buy and sell products, and allow employees of the company to manage the platform. The database stores user information such as login information, addresses, names, user status, orders as well as information about products, such as product names, inventories, reviews, and type of product. The platform allows users to register, search for products. Registered users are provided with a shopping cart, where they can place any number of items from various sellers and check out the items. Registered users can provide and update their name and address, and their password information. Registered users can also apply to become a seller, and sellers can add products and list them on the platform. Seller can also update the listing. Both the buyer and the seller involved in transaction can view the order detail of current and past transactions. Sellers can fulfill the order placed by buyers by shipping the product and providing shipping tracking numbers. The moderator acting as employee are able to accept or deny users requesting an upgrade, and add new departments for sellers to list products under.

Project Design

User types:

Buyer

Each buyer has a shopping cart linked to their account. While browsing the market place, (1) buyers can add product(s) in their shopping cart. Buyers can then access the shopping cart to (2) place order for product(s) of their choosing. In the personal page, buyers can (3) apply to become a seller if the conditions are met, and (4) edit personal information and password information. (5) Buyers can write reviews on products and rate other sellers.

Seller

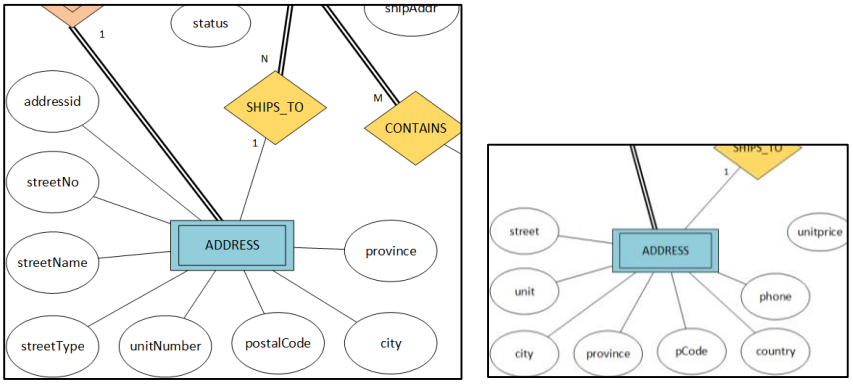
A seller is an upgraded buyer. (1) A seller create a listing which contains newly added product as well as relevant information such as product description, price, quantity available, and department it belongs to. The Seller can then edit or delete the listing(s). The system does security check so that a seller can

edit or delete only their listing. (3) The seller can manage pending orders, where they can approve the orders for shipment to specific buyers, (4) and the seller can access the history of their fulfilled orders.

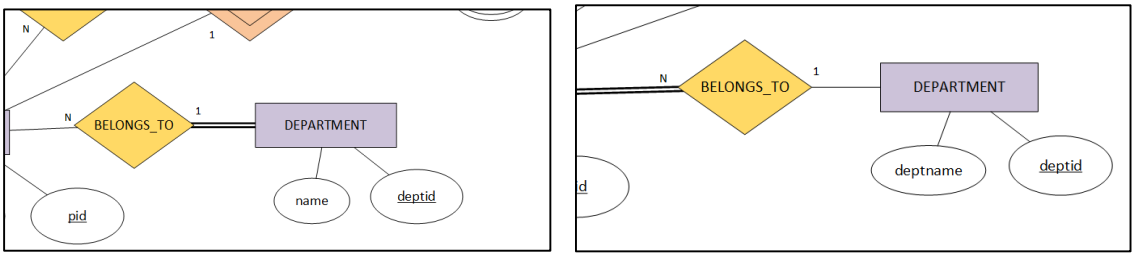
Moderator

Moderators are the overseers of the system. (1) They have the power to approve users who applied to become a seller, and potential employees who want to become moderators. (2) They can also add new departments, to accommodate the variety of products that may exists in the platform.

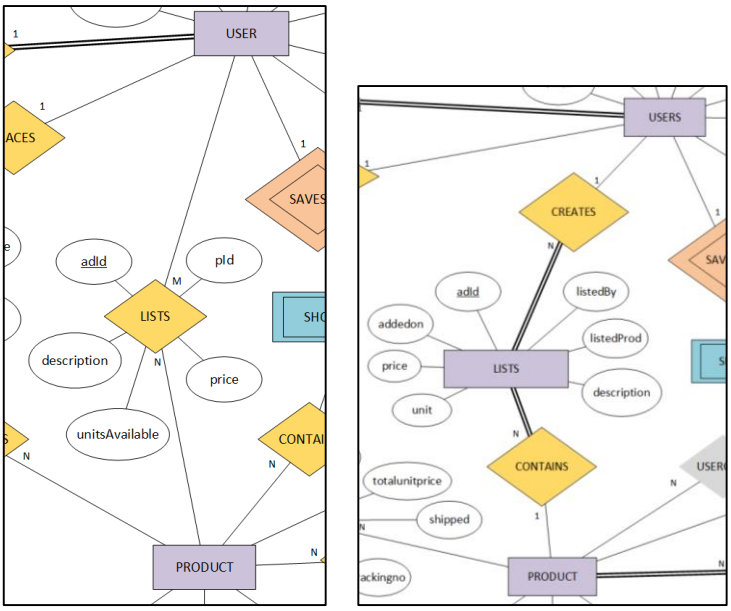
ER



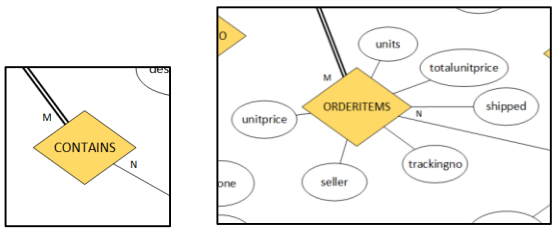
The ADDRESS entity in the original ER diagram was an weak entity and contained 8 attributes – addressId, streetNo, streetName, streetType, unitNumber, postalCode, city, and province where addressId was supposed to be the partial key. In the new iteration of the ER diagram, the ADDRESS entity had following attributes updated - addressId was removed, streetNo, StreetName, streetType, were combined to into the street attribute to reduce complexity, unitNumber was renamed to unit, postalCode was renamed to pCode, and two additional attributes – country, phone were added to accommodate extra level of detail. ADDRESS entity is now directly linked to USERS entity and is identified by USERS.id.



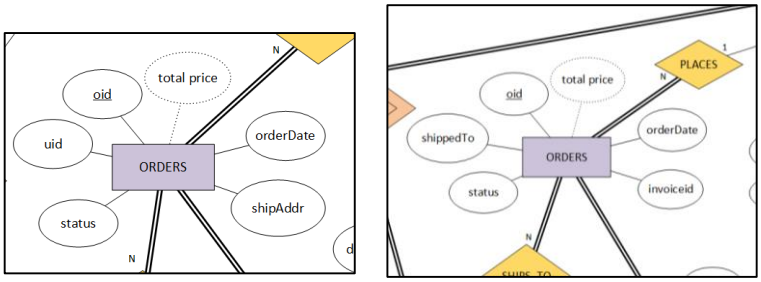
The DEPARTMENT entity in the original ER diagram contained 2 attributes – name and deptid, however in new iteration of the ER diagram, name was renamed to deptname to decrease confusion with other entities with similar attributes while doing join statement. The participation is also changed to reflect the correct relationship between DEPARTMENT and PRODUCT entities.



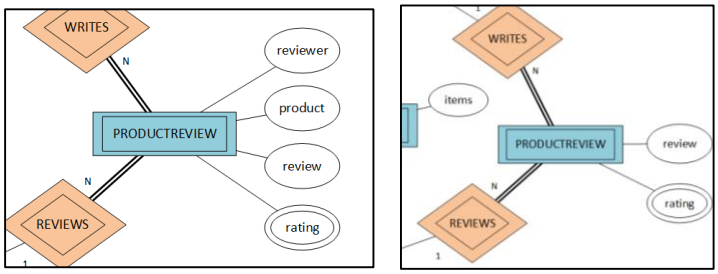
In the original ER diagram, LISTS was a relationship joining USERS to PRODUCT having 5 attributes - adId, description, unitsAvailable, price, and pId. In the new iteration of the ER diagram, LISTS was changed to its own entity type and now contains 7 attributes - adId, listedBy, listedProd, price, description, unit, and addedon, where unitsAvailable in original has been renamed to unit, and pId to listedProd, and two new attributes – listedBy and addedon were added. The decision to change LISTS from a relationship type to an entity type was brought on by the fact that a listing created by a seller required an unique identifier. The LISTS relationship represented an M:N cardinality between USERS and PRODUCT entities, but it was later changed to 1:N cardinality between USERS and LISTS and PRODUCT and LISTS with total participation on LISTS on both relationships.



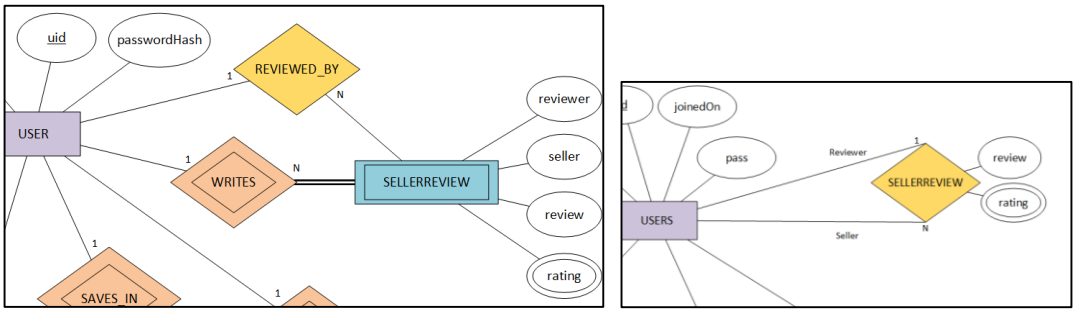
The CONTAINS relationship in original ER diagram was changed to ORDERITEMS in the new iteration of ER diagram. In the original ER diagram, there were no attributes specified for the relationship. It has since changed to include six new attributes – unitprice, seller, trackingno, shipped, totalunitprice, and units.



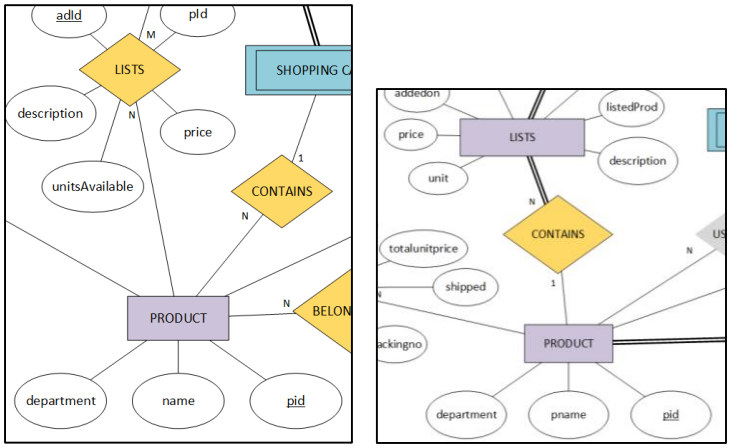
The ORDERS entity in ER diagram contained 6 attributes – oid, uid, orderDate, shipAddr, status, and total price. In the new iterations, uid was removed and shipAddr was renamed to shippedTo. One new attribute was added – invoiceid.



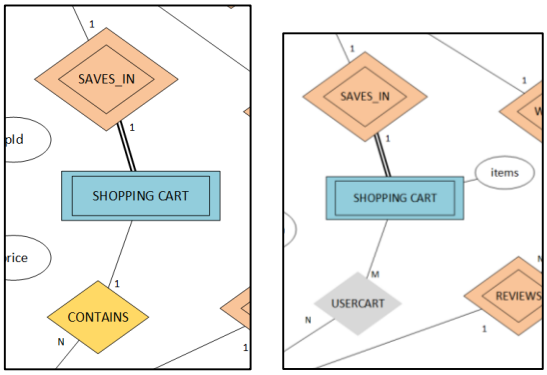
The PRODUCTREVIEW in the ER diagram contained 4 attributes – reviewer, product, review, and rating. In the new iteration, reviewer and product has been removed as they are foreign key referencing USERS.id and PRODUCT.pid respectively.



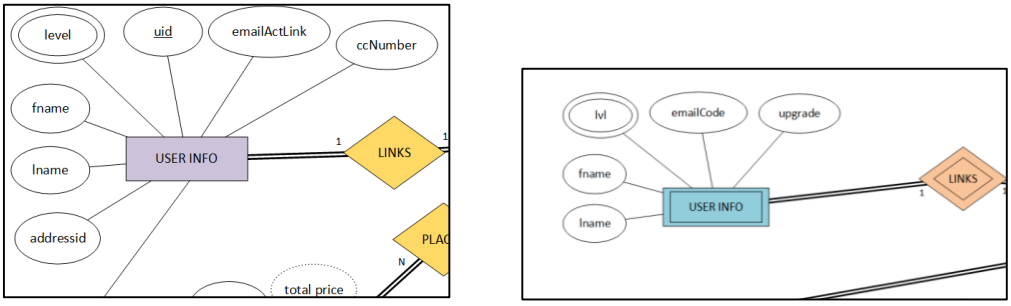
The SELLERREVIEW was an weak entity type in the original ER diagram and contained 4 attributes – reviewer, seller, review, and rating. In the new iteration, SELLERREVIEW was converted from weak entity to a relationship type to illustrate recursive relationship between USERS. It now contains 2 attributes – review and rating.



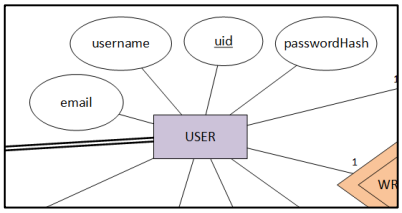
The PRODUCT entity in ER originally contained 3 attributes – pid, department, and name. In the new iteration, name attribute is renamed to pname to avoid confusion in join statement. PRODUCT now has a CONTAINS relationship with LISTS.

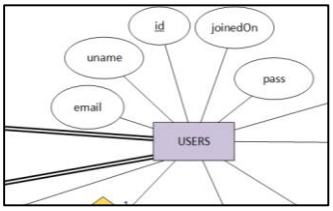


The SHOPPINGCART entity in ER originally contained no attributes. In the new iteration, new attribute was added – items. The CONTAINS relationship in the original ER diagram was a relationship between SHOPPING CART and PRODUCT entities with a 1:N cardinality. The CONTAINS relationship was renamed to USERCART with an M:N cardinality but was later removed, represented by the grey colour, because PHP allows serial array data to be stored as an attribute thus removing the need to create the USERCART relationship in the first place.



The USERINFO entity in the original ER diagram contained 7 attributes – level, uid, emailActLink, ccNumber, fname, lname, addressid. In the new iteration, ccNumber was removed, uid was removed, addressid was removed, and a new attribute, upgrade, was added to specify if the user wants to become a seller, or moderator. In the new iteration USER INFO was changed from a regular to a weak entity as it cannot exist without a user. The LINKS relationship was modified to an identifying relationship to reflect this change. USERINFO entity does not link to ADDRESS entity anymore.

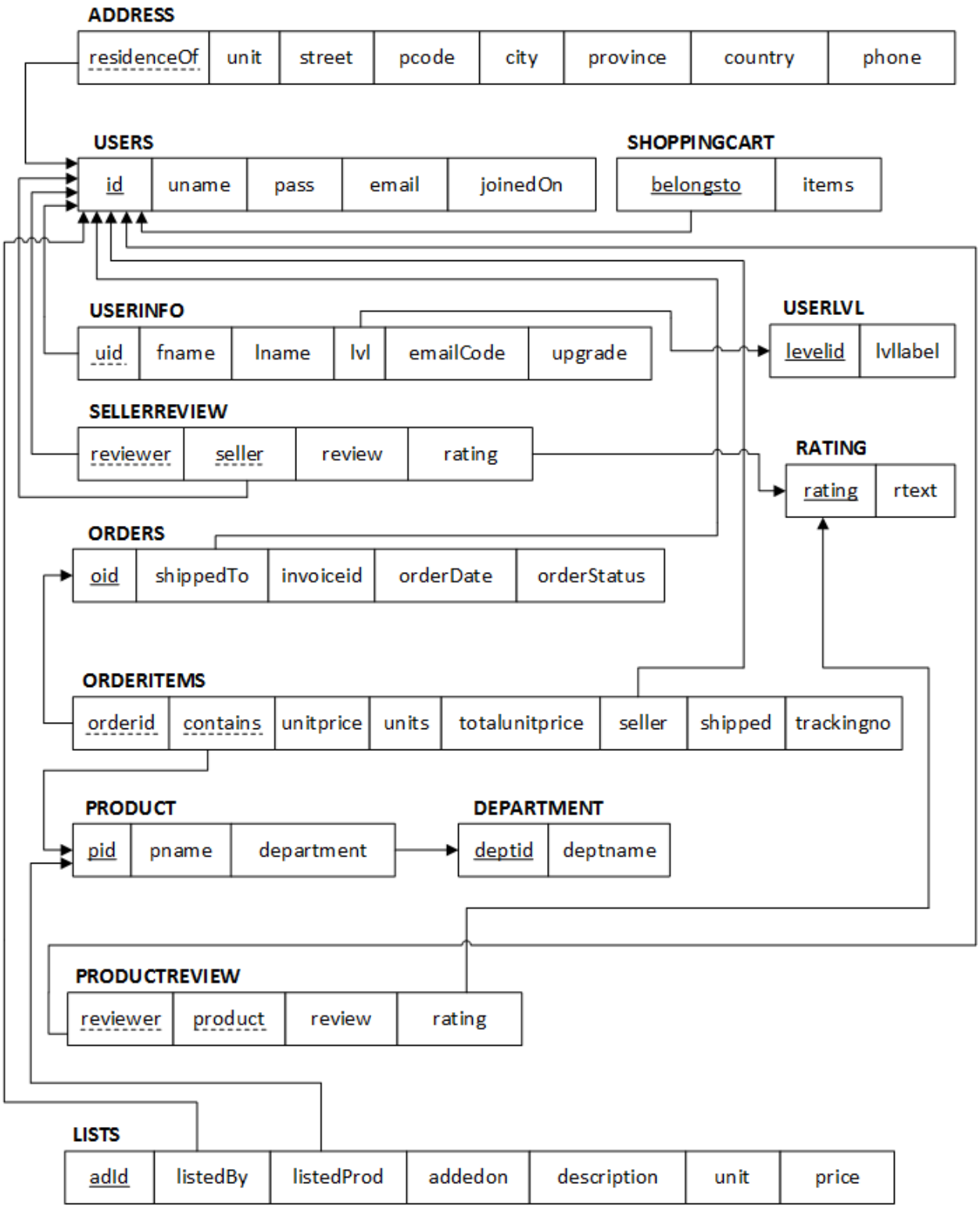




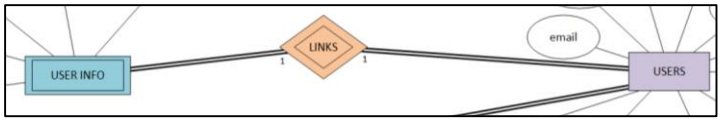
The USER entity in the original ER diagram contained 4 attributes – email, passwordhash, uid, and username. In the new iteration, the joinedDate attribute was added to specify when the user joined the service. Username has since been renamed to uname. passwordhash has been renamed to pass, and uid has been renamed to id. USERS entity now has identifying relationship LIVES_AT which identifies users ADDRESS.

In the original ER diagram, PRODUCTREVIEW.ratings, and SELLERREVIEW.ratings, and USERINFO.level were multivalued attributes of their respective entities. However in the SQL environment, new tables were created for the multivalued attributes.

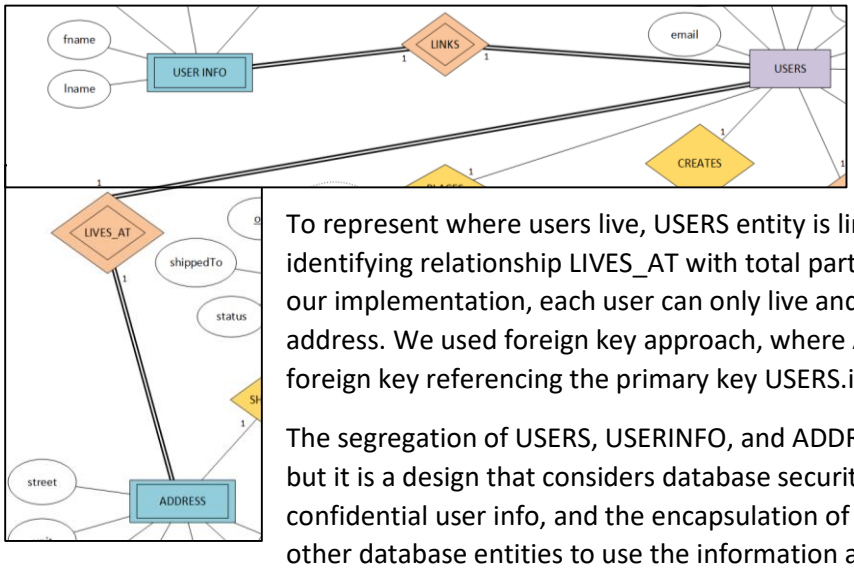
Implementation



The design of this database relied heavily on the ER diagram. The aforementioned ER diagram was used to create a relational schema diagram which was later used for external schema of the production database.

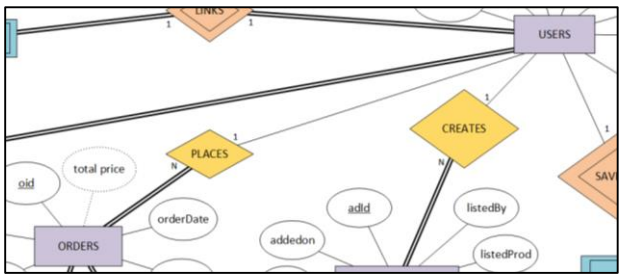


USERS entity has an identifying relationship LINKS with USERINFO entity. The relationship has 1:1 cardinality and has total participation on both side as each user can only have one set of personal information. Since USERINFO is a weak entity, we used foreign key approach, where USERINFO.uid is foreign key referencing the primary key USERS.id.

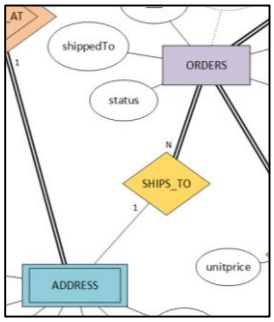


To represent where users live, USERS entity is linked to ADDRESS via 1:1 identifying relationship LIVES_AT with total participation on both side because in our implementation, each user can only live and receive shipments at one address. We used foreign key approach, where ADDRESS.residenceOf is a foreign key referencing the primary key USERS.id.

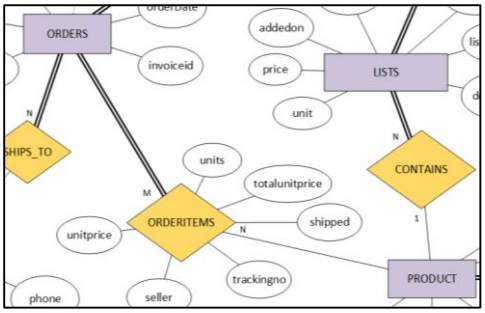
The segregation of USERS, USERINFO, and ADDRESS entity is an unusual design, but it is a design that considers database security, restricting the availability of confidential user info, and the encapsulation of the address information allows other database entities to use the information at ease. The USERINFO is separated from USERS in the event of a query or database malfunction, and/or malicious SQL injection, it will now reveal user's private information such as their names or address. The ADDRESS entity has been encapsulated because USERINFO entity and ORDERS entity refers to the information that is in the ADDRESS, this allows for an easier reference to address information.



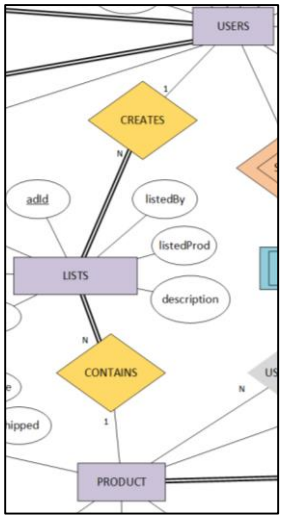
The USERS entity has 1:N relationship PLACES with ORDERS entities, where ORDERS has total participation because USERS may or may not have orders, but orders cannot exists without a user placing it. Given the algorithm, for 1:N cardinality, the entity on the N side – ORDERS, must contain foreign key that references primary key of the entity on 1 side. In this case, ORDERS.shippedTo is a foreign key referencing the primary key USERS.id.



Each ORDERS entity is indirectly linked to an ADDRESS entity via USERS. In the ER diagram, SHIPS_TO relationship is conceptually drawn to signify 1:N cardinality meaning an ORDER can only be shipped to one address but one address can receive many orders. ORDERS.shippedTo is a foreign key referencing USERS.id which is also a foreign key for ADDRESS.residenceOf.



The ORDERS are related to PRODUCT via ORDERITEMS relationship, which is an M:N relationship with total participation on ORDERS side because an order must contain product, but a product may not be contained in any orders. Given the algorithm for M:N relationship, a new table ORDERITEMS was created which contains foreign keys ORDERITEMS.orderid referencing the primary key ORDERS.oid, and foreign key ORDERITEMS.contains referencing the primary key PRODUCT.pid.



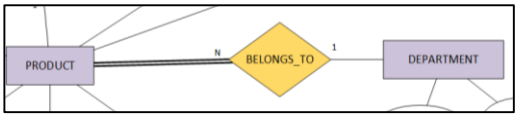
USERS entity is related to LISTS entity via CREATES relationship with 1:N cardinality. Given the algorithm for 1:N relationship, the entity on the N side must contain a foreign key referencing the primary key of the entity on the 1 side. In this case, LISTS.listedBy is a foreign key referencing USERS.id.

PRODUCT entity is related to LISTS entity via CONTAINS relationship wth 1:N cardinality. Given the algorithm for 1:N relationship, the entity on the N side must contain a foreign key referencing the primary key of the entity on the 1 side. In this case LISTS.listedProd is a foreign key referencing PRODUCT.pid.

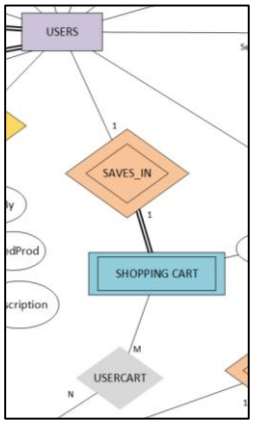
On our platform, user is allowed to upgrade to seller status. Once a user has been upgraded to a seller status, they can add products and listings containing these products. The USERS entity has an 1:N relationship CREATES with LISTS entity. USERS entity has partial participation whereas LISTS entity has total as users may or may not create a listing, but listing cannot exist without a seller. The PRODUCT entity has an 1:N relationship CONTAINS with LISTS entity. PRODUCT entity has partial participation whereas LISTS entity has total as a product may not be listed in a listing, but a listing cannot exist without a product being sold on it.

The product information was segregated from the listings information by creating 2 separate entities called LISTS and PRODUCT because the ORDERITEMS entity needs information about the product to store as order history without re-storing the specific details about the product such as its name and department. If the LISTS and PRODUCT entities were the same, the deletion of a record in this entity would also delete the data about the product itself which would result in a loss of data in the ORDERITEMS table when searching for information about a product in past orders. We decided that

sellers can only edit the listing once a product has been added in the database then and only then can the seller update or delete the listings, but not the product itself.

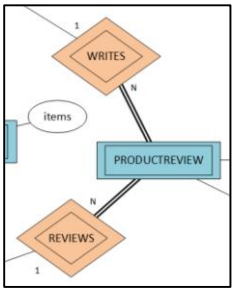


The DEPARTMENT entity is linked to PRODUCT via BELONGS_TO relationship, which has 1:N cardinality and total participation from the PRODUCT entity because a department may or may not contain a product, but a product cannot exists without belonging to a department. Given the algorithm for 1:N cardinality, the entity on the N side must contain a foreign key reference to a primary key of the entity on the 1 side. In this case, foreign key PRODUCT.department references the primary key DEPARTMENT.deptid.



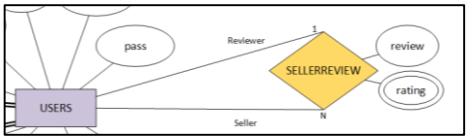
USERS can add PRODUCTS in SHOPPING CART via SAVES_IN relationship. The SAVES_IN relationship has a 1:1 cardinality. Since SHOPPING CART is a weak entity, we used foreign key approach, where SHOPPINGCART.belongsto is the foreign key referencing the primary key USERS.id.

The USERCART relationship between SHOPPINGCART and PRODUCT is a conceptual relationship and has been removed in relational schema denoted by the grey colour, because PHP allows serial array data to be stored as an attribute thus removing the need to create the USERCART relationship in the first place.

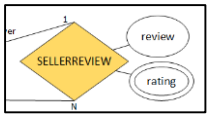


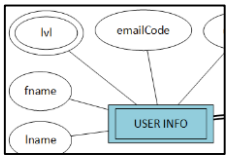
The PRODUCT entity is linked to PRODUCTREVIEW entity via identifying relationship REVIEWS. Given the algorithm for 1:N cardinality, the entity on the N side must contain a foreign key referencing the primary key of the entity on the 1 side. In this case, PRODUCTREVIEW is identified by the foreign keys PRODUCTREVIEW.product which references the primary key PRODUCT.pid and PRODUCTREVIEW.reviewer which is a foreign key referencing the primary key USERS.id. The PRODUCTREVIEW has total participation whereas PRODUCT has partial participation on REVIEWS relationship because a product can have many reviews, but each review is specific to each product.

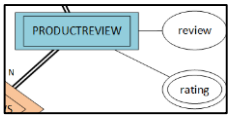
The PRODUCT entity is also linked to USERS entity via identifying relationship WRITES. Given the algorithm for 1:N cardinality, the entity on the N side must contain a foreign key referencing the primary key of the entity on the 1 side. In this case, PRODUCTREVIEW is identified by the foreign key PRODUCTREVIEW.reviewer which is a foreign key referencing the primary key USERS.uid and PRODUCTREVIEW.product which references the PRODUCT.pid.



USERS entity recursive relationship SELLERSREVIEW. The relationship has 1:N cardinality, with 1 on the reviewer side, and N on the seller side. Given the 1:N cardinality algorithm, foreign key approach was used where SELLERREVIEW.reviewer, and SELLERREVIEW.seller are foreign keys both referencing the primary key USERS.id.







USERINFO also contains column determining the status of the users (buyer, seller, moderator). This is done through checking USERINFO.lvl, where USERINFO.lvl is a foreign key referencing the primary key USERLVL.levelid. SELLERREVIEW and PRODUCTREVIEW both contain the rating for review, and this is a multivalued attribute. This attribute is a foreign key referencing primary key RATING.rating.

Implementation: Technical specification

The abundance of database systems that are available on both commercial and open-source markets today, gave us many options to choose from. We considered many databases such as OracleDB, MongoDB, and NoSQL but given our needs and requirement for this project, MySQL database running version 5.7 was the perfect choice. Along with MySQL being a very popular relational database, it has the ability to scale very easily and has many storage engines such as InnoDB, and MyISAM and has support for variety of both primitive and complex data types.

Our project relies heavily on our database and a variety of SQL statements to accommodate the front end operations and GUI. We will discuss some of these important SQL statements in following sections.

These SQL statement will follow chronological timeline of user's experience on the platform.

Starting with sign up process, the registration code checks to see if the user already exists in the system and does so by using the following query:

```
SELECT * FROM `USERS` WHERE `uname` = `$uname`;
```

After this test is passed, and amongst other input sanitization, the user is created in our database. This affects three database tables using the following queries:

```
START TRANSACTION;
INSERT INTO `USERS` ("uname", "pass", "email", "joinedOn") VALUES ("$this->vuname",
"$this->hashpassword", "$this->vemail", $this->dtime);
INSERT INTO `ADDRESS` ("residenceOf") VALUES ($newuserid);
INSERT INTO `USERINFO` ("uid", "lvl", "emailCode") VALUES ($newuserid, 1, "$this-
>emailCode");
COMMIT
```

Once the user is created in the database, user can then log in and our log in code performs the following queries:

```
SELECT `id`, `uname`, `pass`, `email` FROM `USERS` WHERE `uname` = $this->user;
```

The user can search for listings in the market place, where they can search by keywords in product name and product description. This is done through following query:

```
SELECT * FROM `MasterDB`.`LISTS` JOIN `PRODUCT` ON `LISTS`.`listedProd` = `PRODUCT`.`pid`
JOIN `DEPARTMENT` ON `PRODUCT`.`department` = `DEPARTMENT`.`deptid` WHERE
`description` LIKE `%$_POST["search"]%` OR `pname` LIKE `%$_POST["search"]%`;
```

The main page of the website lists all the active listings, which are retrieved via following query:

```
SELECT * FROM `MasterDB`.`LISTS` JOIN `PRODUCT` ON `LISTS`.`listedProd` = `PRODUCT`.`pid`
JOIN `DEPARTMENT` ON `PRODUCT`.`department` = `DEPARTMENT`.`deptid`;
```

The product page lists the information of the product such as seller, quantity available, price, and product description. This data is pulled using the following query:

```
SELECT * FROM `LISTS` JOIN `PRODUCT` ON `LISTS`.`listedProd` = `PRODUCT`.`pid` JOIN `DEPARTMENT` ON `PRODUCT`.`department` = `DEPARTMENT`.`deptid` WHERE `adId` = $this->listId;
```

The user's shopping cart is controlled by three SQL queries, which is used to retrieve the shopping cart, update the cart with new product, and clear the cart:

Retrieve the cart:

```
SELECT * FROM `SHOPPINGCART` WHERE `SHOPPINGCART`.`belongsto` = $this->getuid();
```

Update the cart:

```
UPDATE `SHOPPINGCART` SET `items` = $cdata WHERE `belongsto` = $this->getuid();
```

Clear the cart:

```
UPDATE `SHOPPINGCART` SET `items` = NULL WHERE `belongsto` = $this->getuid();
```

Once the item is in the shopping cart, user can check out the item(s) which triggers a host of queries:

```
START TRANSACTION;
INSERT INTO `ORDERS` (`uid`, `invoiceid`, `orderDate`, `shippedTo`, `ordstatus`) VALUES ($this->getuid(), randomNumber(6), `date("Y-m-d H:i:s")`, $this->getUserData("id"), `Placed`);
// loop to insert all items in the order
INSERT INTO `ORDERITEMS` (`orderid`, `contains`, `unitprice`, `units`, `totalunitprice`, `seller`) VALUES ($orderid, $i->getProdId(), $i->getPrice(), $post[$item], $i->getPrice() * $post[$item], $i->getSellerId());
…
COMMIT;
```

The user and sellers can check the orders that they are part of in the orders details page:

```
SELECT * FROM `ORDERITEMS` JOIN `ORDERS` ON `ORDERITEMS`.`orderid` = `ORDERS`.`oid`;
```

The user is able to change their personal information such as their name, address, and password using the following queries:

```
UPDATE `USERINFO` SET `fname` = `$this->vfname`, `lname` = `$this->vlname` WHERE `USERINFO`.`uid` = $this->getuid();
UPDATE `ADDRESS` SET `street` = `$this->vstreet`, `unit` = `$this->vapt`, `city` = `$this->vcity`, `province` = `$this->vstate`, `pcode` = `$this->vpcode`, `country` = `$this->vcountry`, `phone` = `$this->vphone` WHERE `ADDRESS`.`residenceOf` = $this->getuid();
UPDATE `USERS` SET `pass` = `$hashpassword` WHERE `USERS`.`id` = $this->uid;
```

The sellers are able to retrieve orders that contains product(s) they are selling:

```
SELECT * FROM `ORDERITEMS` JOIN `ORDERS` ON `ORDERITEMS`.`orderid` = `ORDERS`.`oid` WHERE `seller` = $this->getuid() AND `shipped` = `N`;
```

The sellers can fulfill a placed order by providing a shipping number for each of their items in the order:

UPDATE `ORDERITEMS` SET `trackingno` = `$tid`, `shipped` = `Y` WHERE `orderid` = oid AND `contains` = pid;

Sellers are also able to add or update their product listing which use the following queries:

START TRANSACTION; *(add Product)*
// first query
INSERT INTO `PRODUCT` (`pname`, `department`) VALUES($this->vpname, $this->vdeptid);
// second query
INSERT INTO `LISTS` (`listedBy`, `listedProd`, `price`, `description`, `units`, `addedon`) VALUES ($this->getid(), $newProdID, $this->vprice, $this->vdesp, $this->vqty, $this->dtime);
COMMIT;

START TRANSACTION; *(update listing)*
SELECT `listedProd` FROM `LISTS` WHERE `adId` = $IID;
// first query
UPDATE `PRODUCT` SET `pid` = `$pID["listedProd"];
// second query
UPDATE `LISTS` SET `adId` = `$IID`;
COMMIT;

The platform moderators are able to update and approve or deny user requests to become a seller, which use the following queries:

START TRANSACTION;
SELECT * FROM `USERINFO` WHERE `lvl` = 1 AND `upgrade = `Y`;
(approve) **UPDATE `USERINFO` SET `lvl` = $lvl+1, `upgrade` = `N`, WHERE `uid` = $q[`uid`];**
OR
(deny) **UPDATE `USERINFO` SET `upgrade` = `N` WHERE `lvl` = $lvl;**
COMMIT;

They are also able to update and approve or deny other users who have requested to become a moderator, which use the following queries:

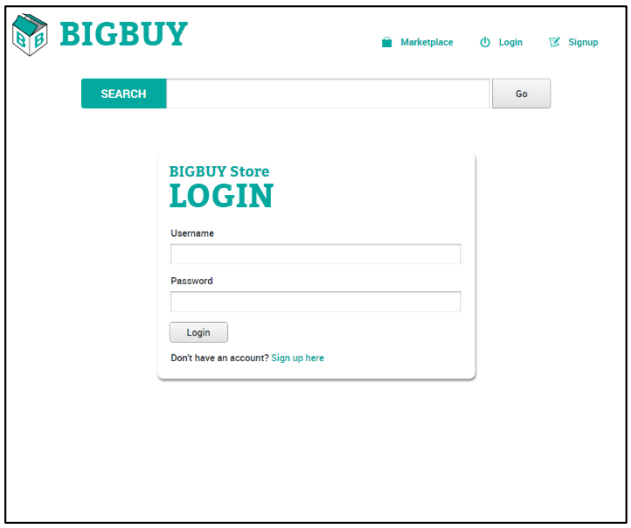
START TRANSACTION;
SELECT * FROM `USERINFO` WHERE `lvl` = 2 AND `upgrade` = `Y`;
(approve) **UPDATE `USERINFO` SET `lvl` = $lvl+1, `upgrade` = `N`, WHERE `uid` = $q[`uid`];**
OR
(deny) **UPDATE `USERINFO` SET `upgrade` = `N` WHERE `lvl` = $lvl;**
COMMIT;

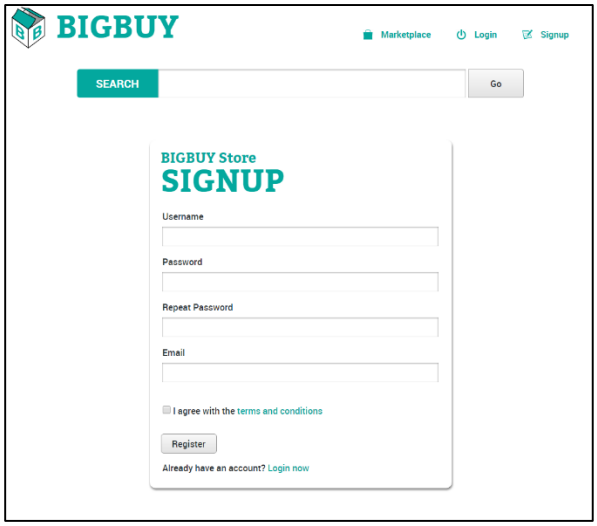
Finally, moderators can add new departments to the marketplace where items can be added to:

INSERT INTO `DEPARTMENT` (`deptname`) VALUES ($this->vdeptname);

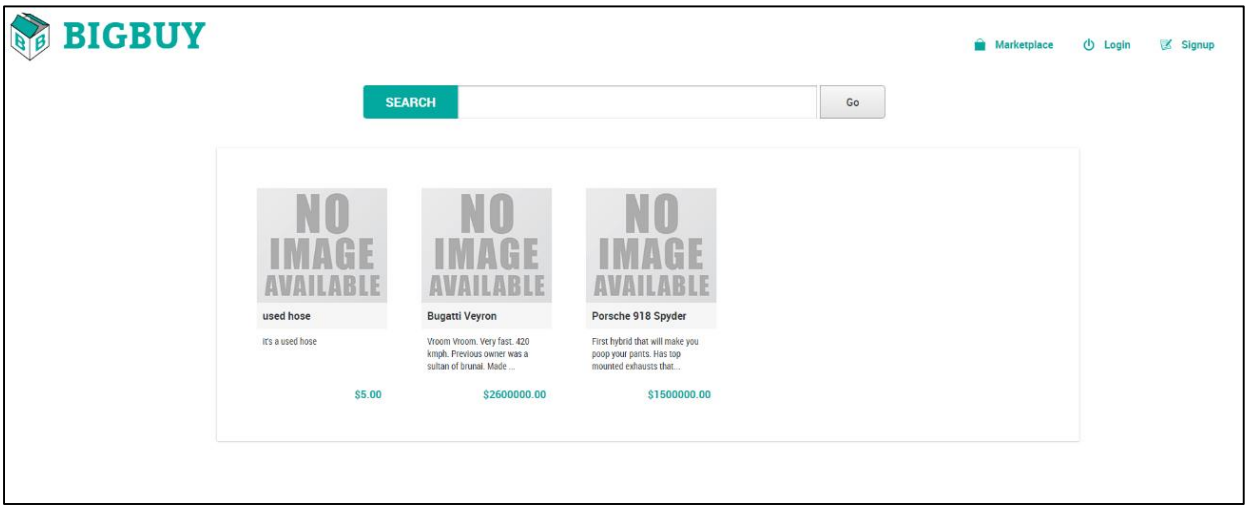
Implementation: User Interface

The platform has many different user interfaces to accommodate the three different types of users. Here is a brief description and demonstration of how they work. There are login and register pages to allow users to login to their accounts or signup to get a new account.



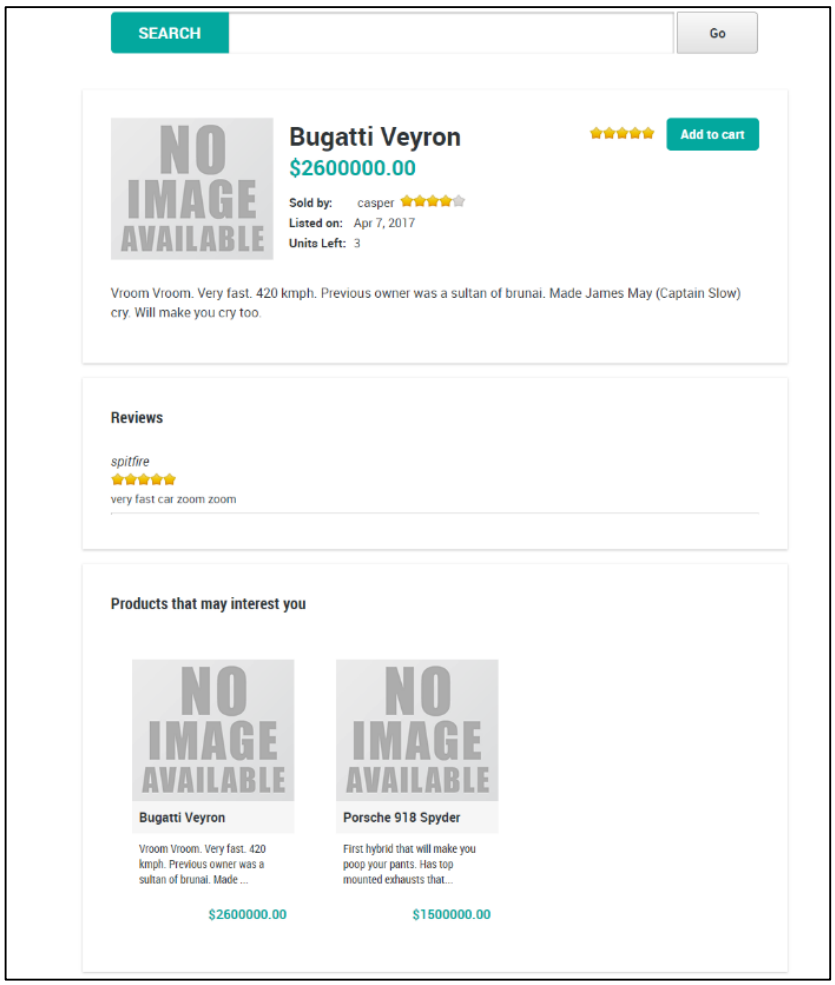


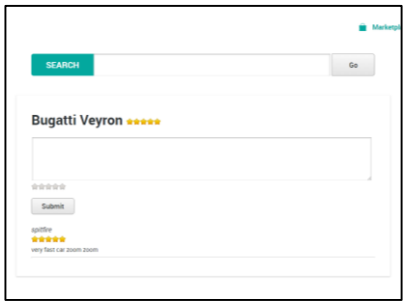
Both non-registered and registered users can access the marketplace where listed products can be seen.

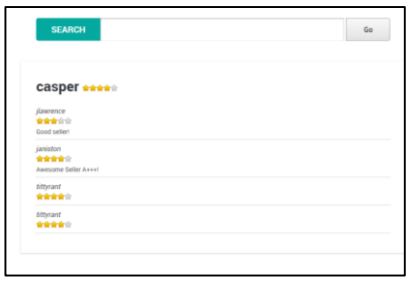


Registered users as well as visitors to the website can also search for products by keyword by entering at the permanently fixed search bar at the top of the marketplace.

Each product listing has its own product page where interested buyers can see more details such as the seller's name, their rating, the product's rating, how many units are left, and a list of other potential interesting items from the same department. Users can add the item to their shopping cart if they would like to purchase it.

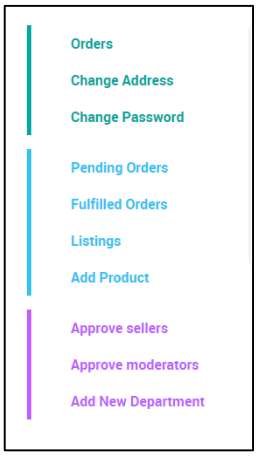


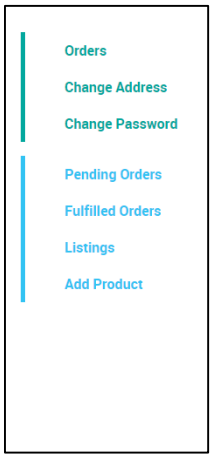


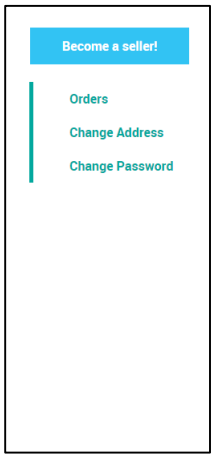


The potential buyer can also click on the stars beside the username and add to cart to view the user ratings and product ratings to see if they are buying from a reputable source.

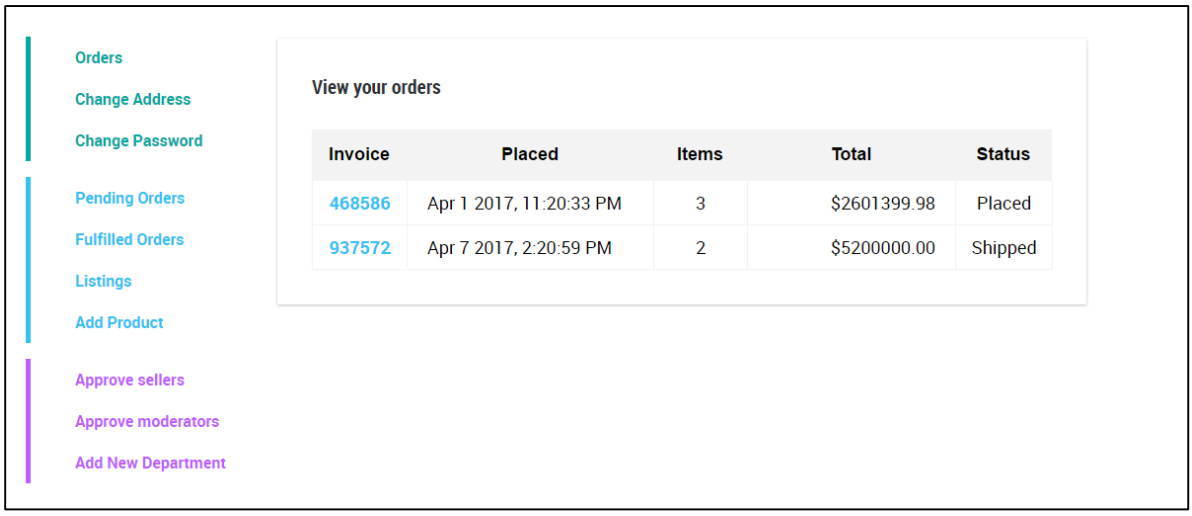
The secure areas of the website are customized based on the status of the users. The options available to users, sellers, and moderators are different. The left menu items differ in the following ways:



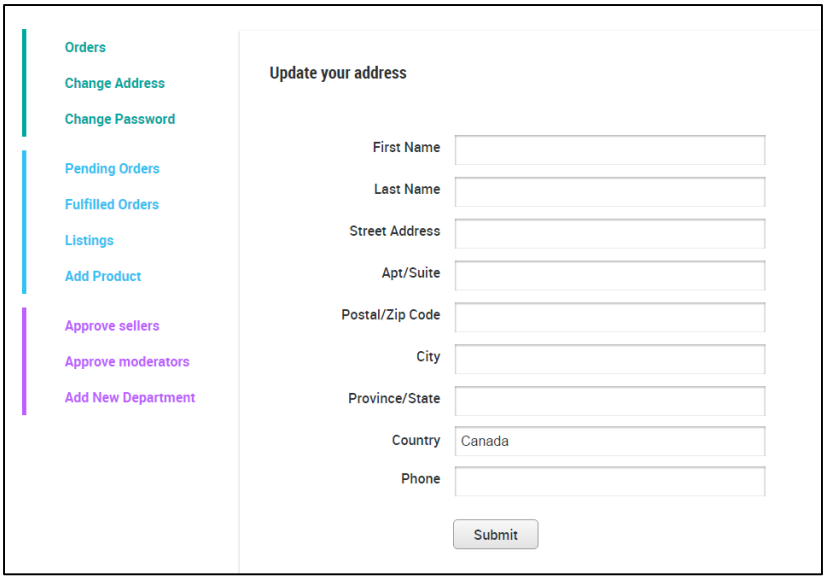


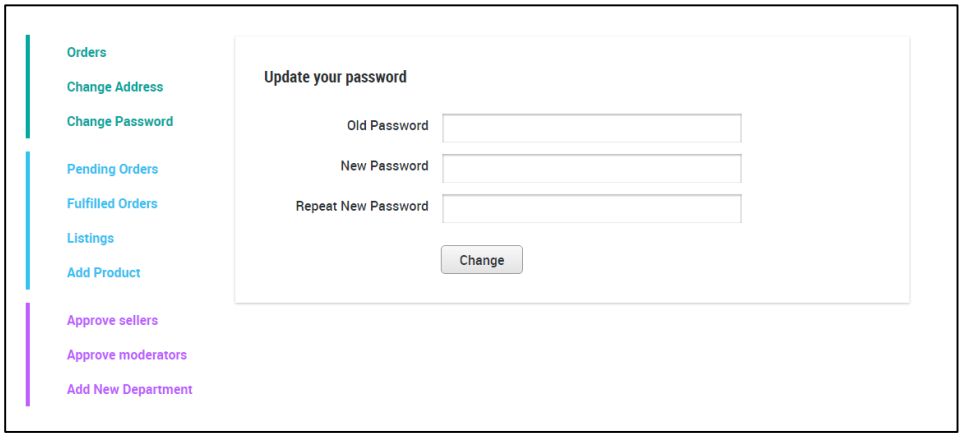


Buyers are able to view their previous orders.

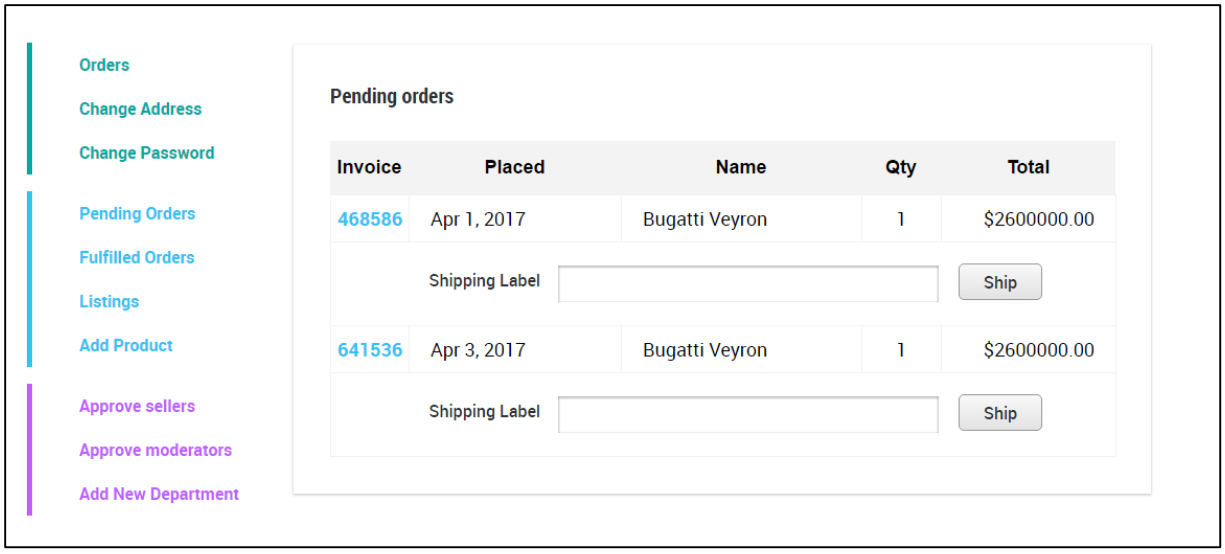


Users are also able to change their address information and password information.

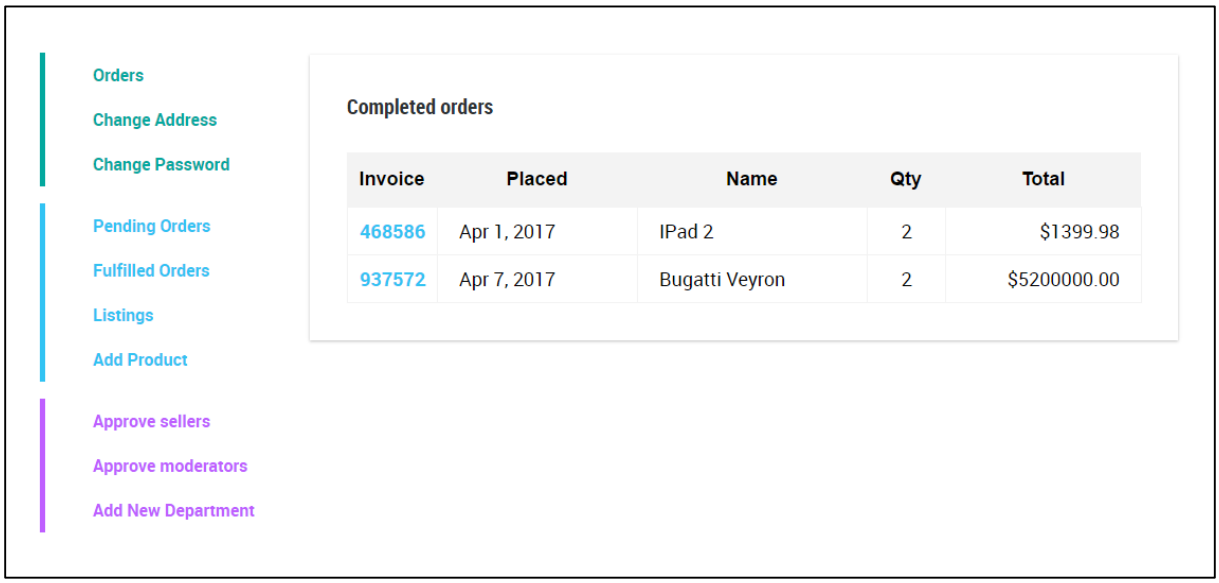




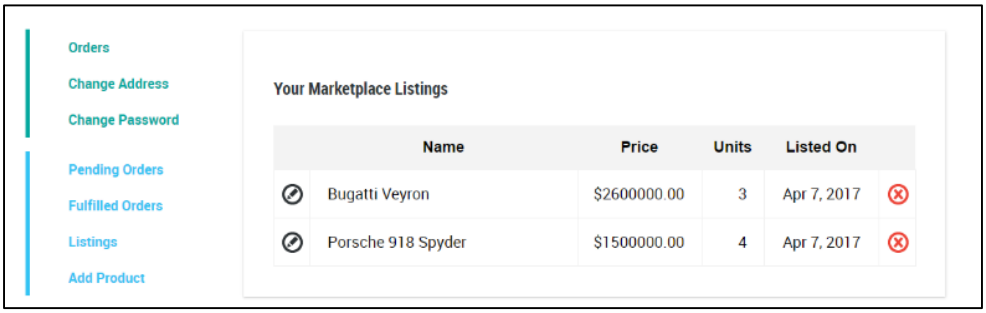
The sellers have an extra dimension of features available to them. They are able to view their pending orders where they can enter the Shipping tracking number for the products that need to be shipped which means they are fullfilling the order.



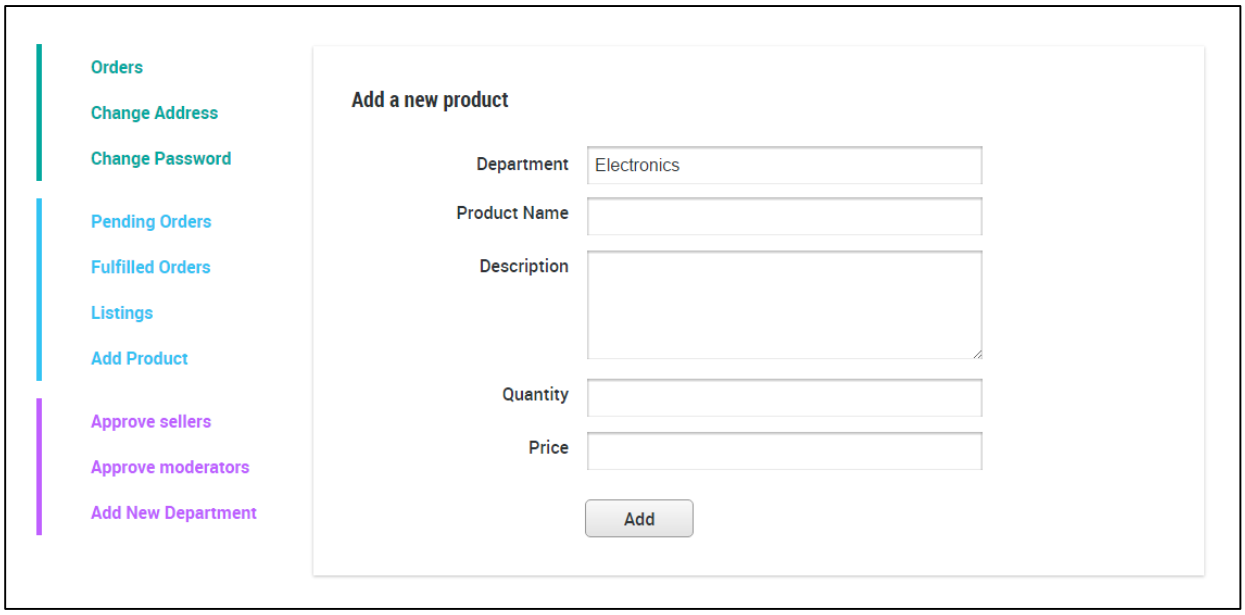
They can also view their fulfilled and completed orders.

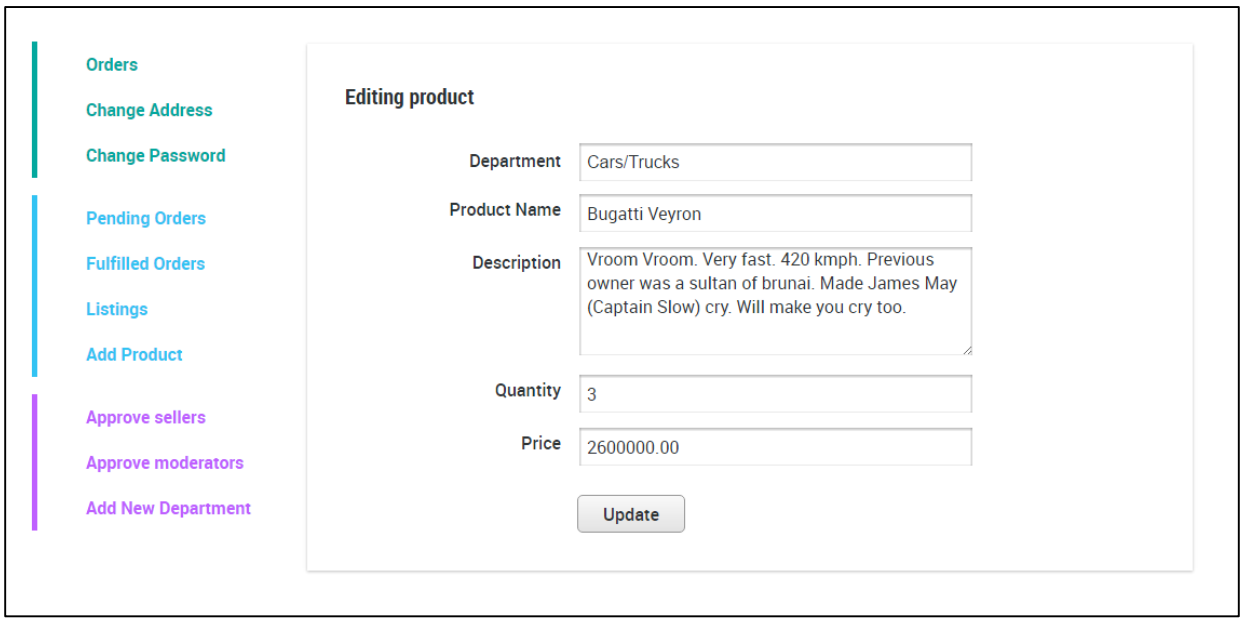


In addition to that, the users can also view the ads that they have listed on the market place.

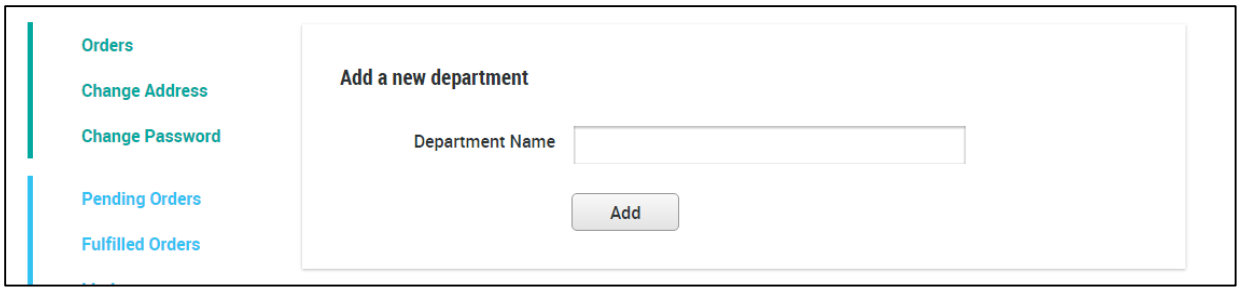


Users can also edit and add new products by clicking the Add Product or the Edit symbol on their listing.

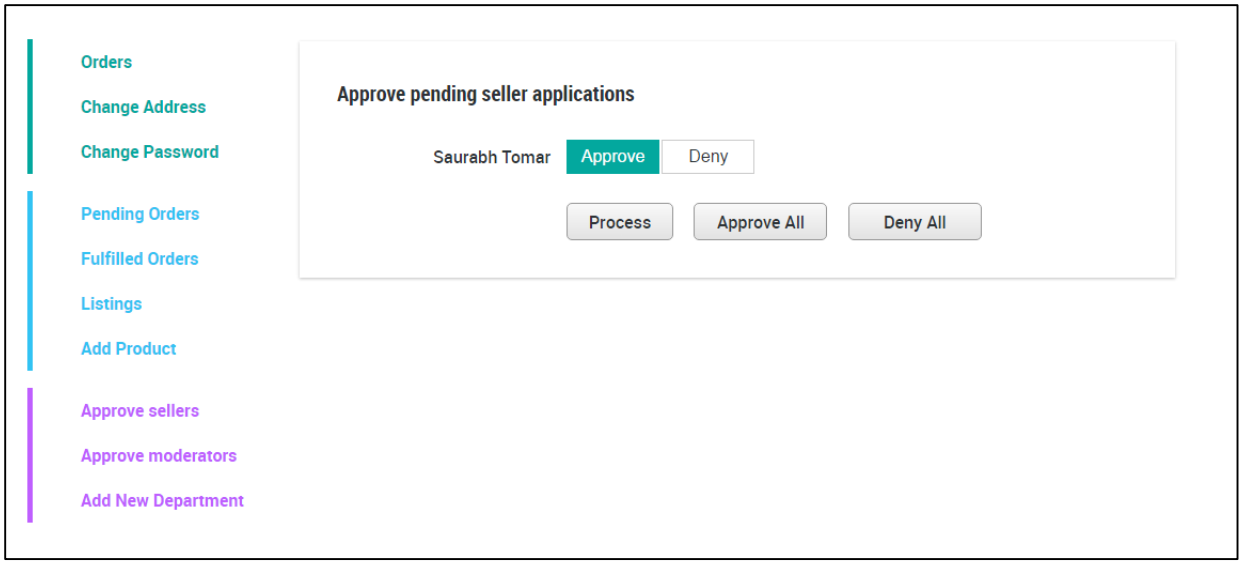


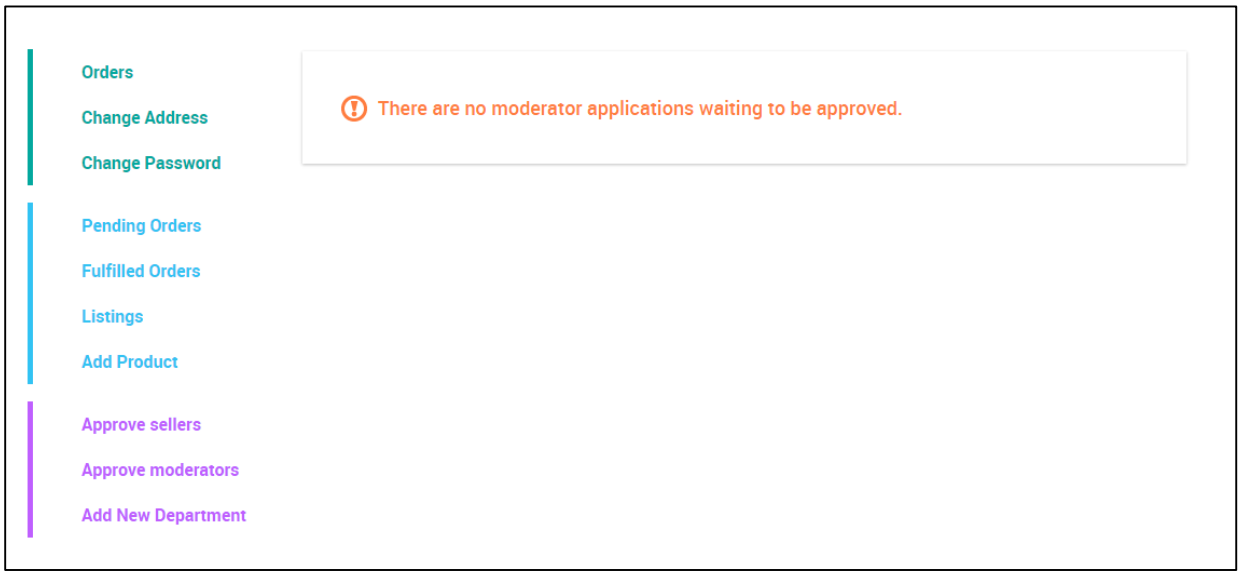


The next level of users are the moderators who can add a new item Department for sellers.

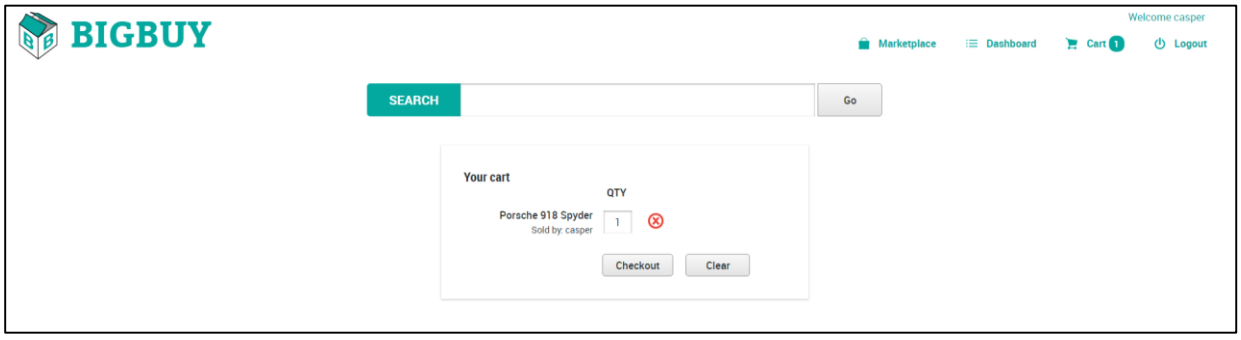


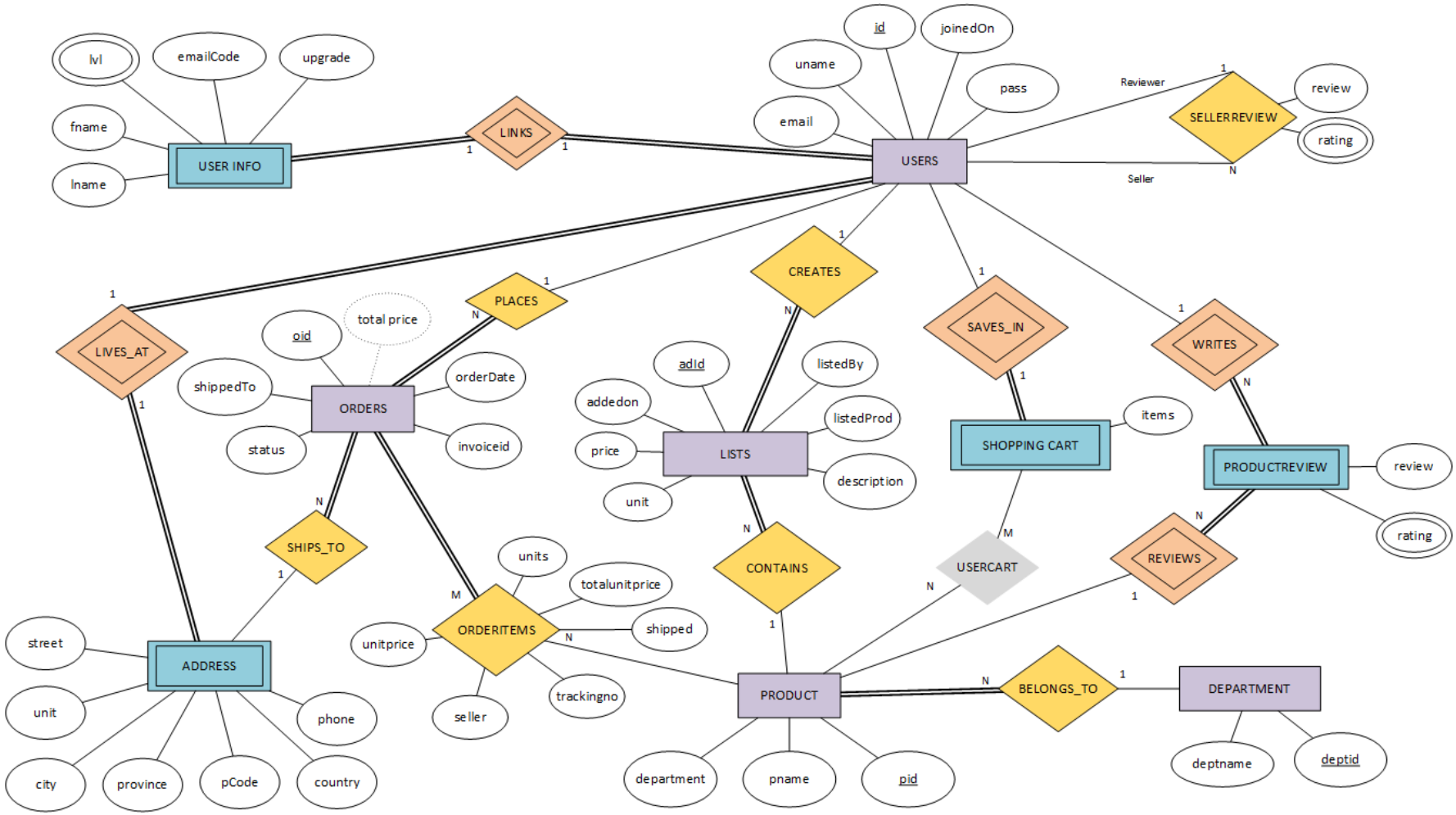
The moderators are also in charge of approving applications that regular users have submitted to become sellers and users that have applied to become moderators. They can also approve all and deny all users in one go as well.





All the registered users are able to add and remove items to their carts, change quantity, and checkout.




lvl
emailCode
upgrade
fname
lname
USER INFO
LINKS
id
joinedOn
uname
pass
email
USERS
Reviewer
SELLERREVIEW
review
rating
Seller
LIVES_AT
PLACES
CREATES
SAVES_IN
WRITES
oid
total price
shippedTo
orderDate
status
invoiceid
ORDERS
adId
listedBy
addedon
listedProd
price
LISTS
description
unit
items
SHOPPING CART
PRODUCTREVIEW
review
rating
SHIPS_TO
CONTAINS
USERCART
REVIEWS
units
totalunitprice
unitprice
ORDERITEMS
shipped
trackingno
seller
street
ADDRESS
unit
phone
city
province
pCode
country
PRODUCT
BELONGS_TO
DEPARTMENT
department
pname
pid
deptname
deptid