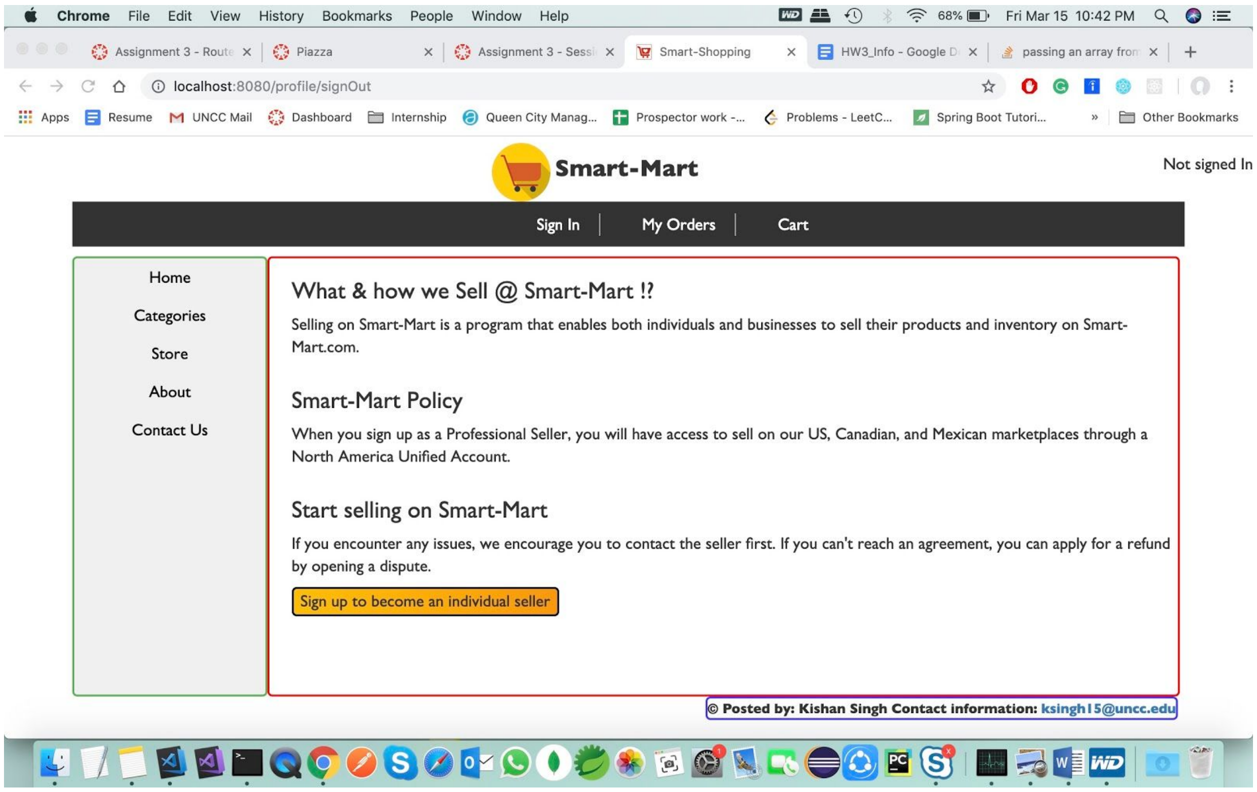
Assignment - 3

****Note : If running the sql scripts throws syntactical error, consider using the script stored in /models/dummy.js**
This file is same as hw4_create_db**

Important updates in this assignment : UserProfileDB.js

- New functions have been added in the userProfileDB.js with new connections to the Database via mongoose models.
- Promises have been used to handle databases queries since they run in an asynchronous fashion.

HomePage



What & how we Sell @ Smart-Mart !?

Selling on Smart-Mart is a program that enables both individuals and businesses to sell their products and inventory on Smart-Mart.com.

Smart-Mart Policy

When you sign up as a Professional Seller, you will have access to sell on our US, Canadian, and Mexican marketplaces through a North America Unified Account.

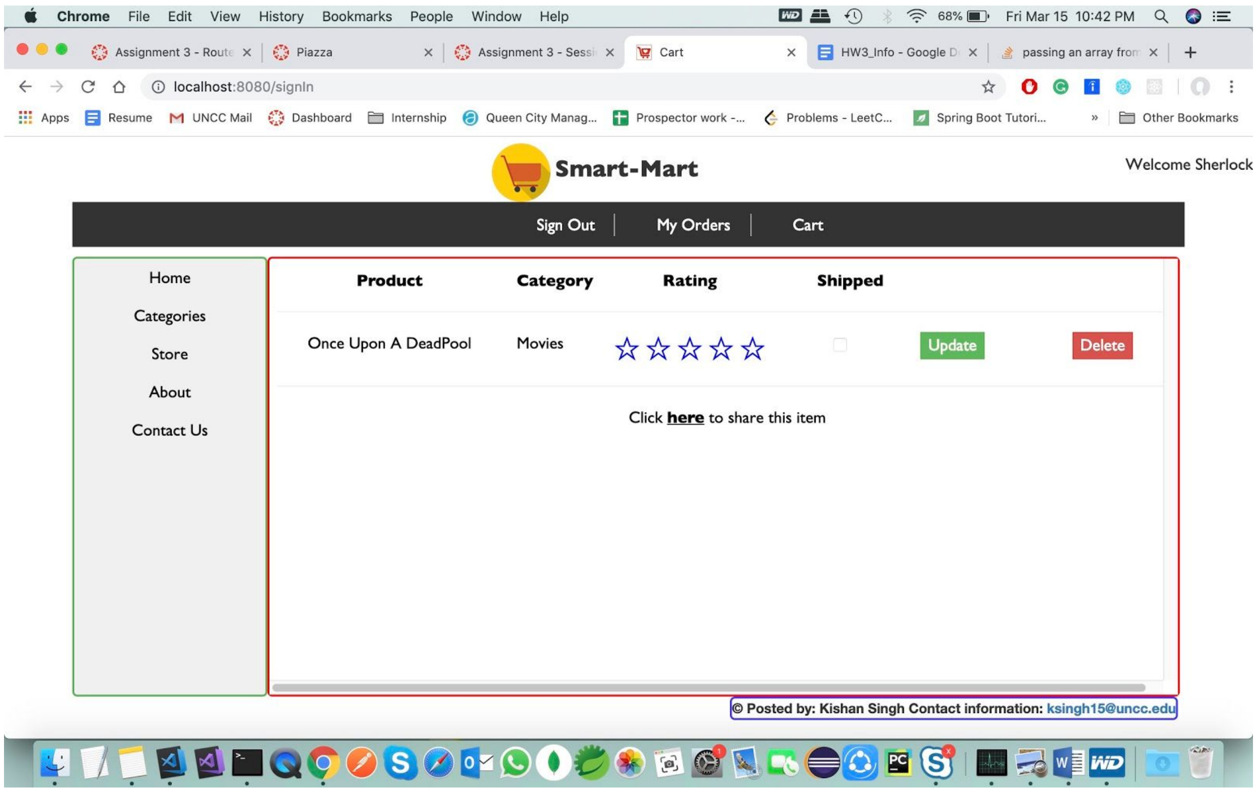
Start selling on Smart-Mart

If you encounter any issues, we encourage you to contact the seller first. If you can't reach an agreement, you can apply for a refund by opening a dispute.

Sign up to become an individual seller

© Posted by: Kishan Singh Contact information: ksingh15@uncc.edu

myItems Page



Smart-Mart

Welcome Sherlock

Sign Out | My Orders | Cart

Home

Categories

Store

About

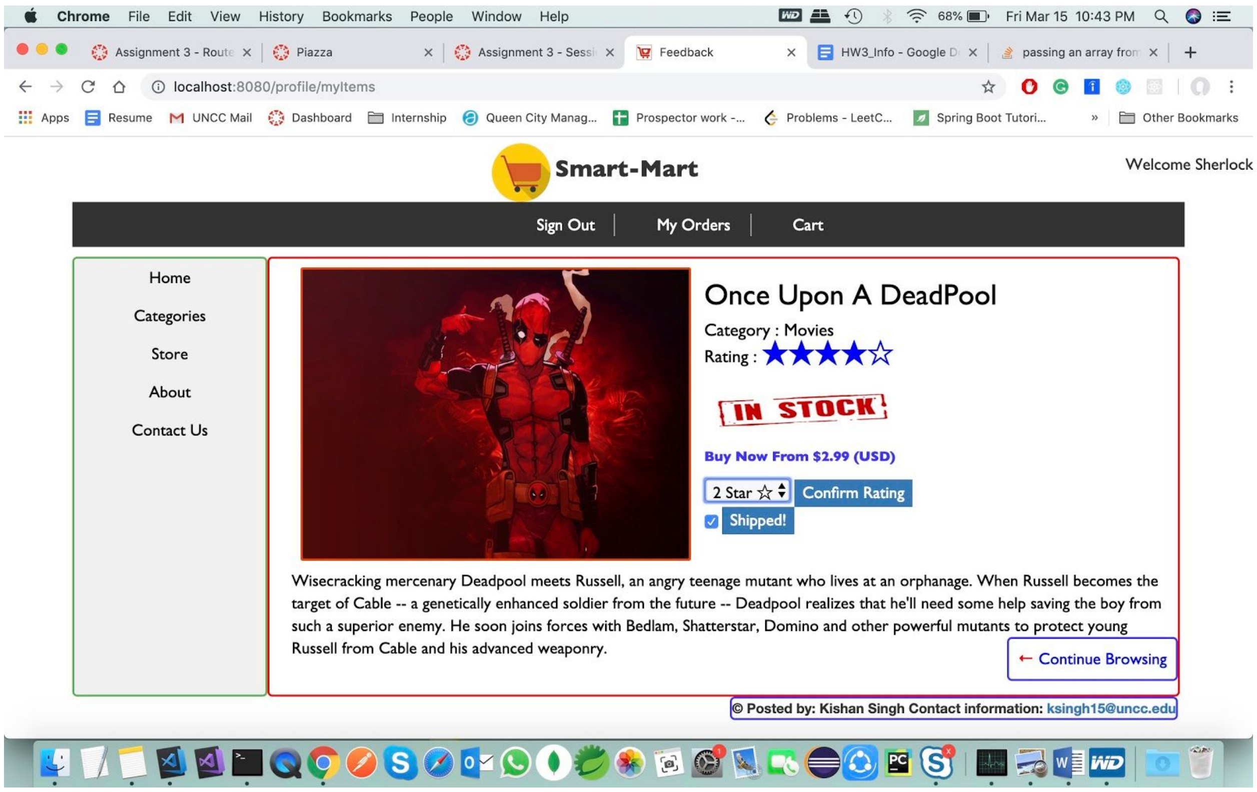
Contact Us

Product	Category	Rating	Shipped		
Once Upon A DeadPool	Movies	☆☆☆☆☆	☐	Update	Delete

Click **here** to share this item

© Posted by: Kishan Singh Contact information: ksingh15@uncc.edu

Item feedback page



Smart-Mart

Welcome Sherlock

Sign Out | My Orders | Cart

Home

Categories

Store

About

Contact Us

Once Upon A DeadPool

Category : Movies

Rating : ★★★★☆

IN STOCK

Buy Now From $2.99 (USD)

2 Star ☆ Confirm Rating

✓ Shipped!

Wisecracking mercenary Deadpool meets Russell, an angry teenage mutant who lives at an orphanage. When Russell becomes the target of Cable -- a genetically enhanced soldier from the future -- Deadpool realizes that he'll need some help saving the boy from such a superior enemy. He soon joins forces with Bedlam, Shatterstar, Domino and other powerful mutants to protect young Russell from Cable and his advanced weaponry.

← Continue Browsing

© Posted by: Kishan Singh Contact information: ksingh15@uncc.edu

updateItem functionality has been split into two :

- **updateItemRating** : to update userItem ratings.
- **updateItemFlag** : to update madeIt status of userItems.

This controller handles all user activities from adding items, removing,getting them rated and marking shipping status for each of them.
It also takes care of all validation in a responsive manner.

myItems.ejs has two forms - one for updating rating of the items and other one for deleting items.It also has a dynamic checkbox to view the shipment status of the particular item.

Feedback.ejs has two forms - one for updating the item rating and the other one to update the shipment status of the item via a checkbox.

Sign In and Sign Out:

- SignIn button on the left side of user-navigation puts the user in the session.
- SignOut Button on the user navigation removes the user from the session and renders the homePage.
- For signIn and signOut, I have not created action parameters, rather chosen to create separate controllers.

ProfileController has action parameters :

1. **Save** : to save item from item.ejs to myitems.ejs
2. **deleteItem** : to delete item from myitems.ejs
3. **updateProfile** : to navigate to the feedback page and render the feedback form to update the rating of the item or it's madeIt status
4. **updateRating** : used when user attempts to change the rating of the item from the feedback page and navigates to the myitems page when user clicks confirm rating.
5. **updateFlag** : used when user has finalised the madeIt status on the feedback page.When the user clicks the madeIt button after checking/unchecking the checkbox on feedback page, the changes are reflected on the myitems.ejs

Feedback page has been made dynamic.It reflects the values from the userProfile inside it's ejs and changes made on the page remain consistent even when navigated back and forth among other pages.

Session

request.session.theUser maintains the state of the user login information at every stage and is used for validation and representation purposes.
If user is not logged in, and he traverses to the item page and tries saving it, then the page redirects you to the categories page.

Dialog boxes to display the error/warning messages:

1. If user tries saving an item without having logged in, view lands on categories page showing list of items and displays a dialog showing the reason of redirect.
2. If user tries rating an item without having logged in, view lands on categories page showing list of items and displays a dialog showing the reason of redirect.
3. If user signs out, view is redirected to home showing a dialog message without an appropriate response message.
4. If user tries adding an item that already exists in his/her savedList, a dialog box with appropriate message is shown.

Security :

Values have been sent with post methods containing key values pairs that are **hidden** so that user has no control over which item to perform the operation on.

Secondly, the itemList is sent along with all post forms to keep track of items that are in the list and to apply updates/deletions on those items only.Comparisons with the itemList and the specific item on which operations are performed will ensure that we keep clean the system from interruptive attacks.
For this purpose an itemList has been prepared from the request.session.userProfile list.
This itemList at any point of time will contain a list of itemCode values that will be present in the userItems/request.session.userProfile.
When these values get submitted from the post method via forms, they will be validated on the controller side and response will be rendered appropriately.
The above logic has been implemented for the 4 action parameters :

1. **Save** - checked to see if the item being added is already present in list and an appropriate dialog box is shown.

2. **updateRating** - checked to see if user/hacker is attempting to modify values other than the user interface.
3. **updateFlag** - checked to see if user/hacker is attempting to modify values other than the user interface.
4. **updateProfile** - checked to see if user/hacker is attempting to modify values other than the user interface.

Challenging parts were:

- **Mongoose models** : deciding on the final efficient model was a challenging part.There were many ways to handle the userProfile model.I handled it by storing (userId, UserItem) pairs and retrieving them based on the userId via mongoose models.
- **Replacing old data retrievals** with the new database connection queries and maintaining data integrating was essential.
- **Maintaining the right state of data** throughout the application was challenging.
- **Handling promises for asynchronous data fetching** failing which previous/uninitialized state of data would be carried ahead and fail validations.

--