

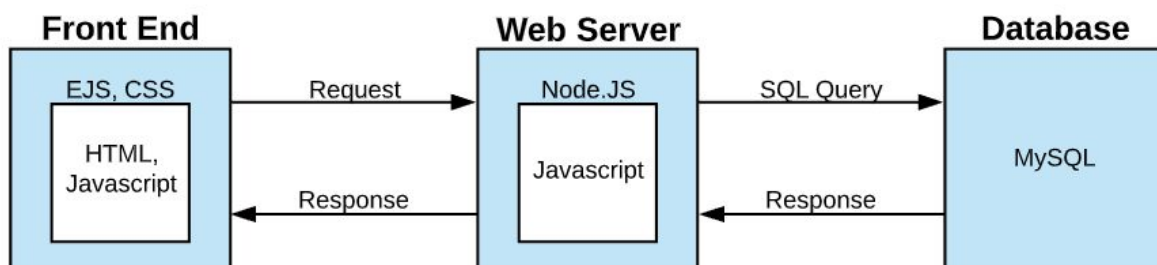
FlipKart
Danny Song, Ting Ting Xu, Jin-Han Han
Team 3
CS157a Section 1

Project Description:

This project will create a worldwide online shopping website to support e-commerce service. It provides an electronic retail and business platform for registered users to purchase and sell products. This electronic platform will replace the typical online shopping website. It will provide small business or individuals an easier way to post and sell their products, and also serves buyers a more fast and simple way to finish payment transaction. It will become the mainstream online shopping system in the United State and overseas The stakeholders will be 77 billion people worldwide, those people including retail suppliers, entrepreneurs. This platform will provide users a simple, fast, and secure e-commerce site.

System Environment:

This ecommerce site will be designed to run within the clients browser and thus the GUI will be created using HTML(EJS), CSS, and Javascript. The web server will be handled using Node JS and thus will be developed in Javascript. This Node JS server will provide the client with the front end as well as connect them to the database which will use MySQL as its database management system. The Node server will communicate with the MySQL database through the Node JS webservice.



Hardware and Software Used:

- Three Tier Environment
 - MySQL version 14.14 Database Management System
 - Node JS Express web server
 - EJS(HTML w/ embedded JS), CSS client side
- Application Languages
 - Javascript, HTML, CSS,

Functional Requirements:

Users of our software will be buyers of products. They will be able to access our system through their browsers and log in to their own personal accounts. This account will keep track of the user's personal information such as name, shipping address, billing address, and payment information. As users, they will have CRUD functionality for this information. Each account will also keep track of information such as their current shopping cart.

The main feature of the application allows users to buy products. They will be able to interact with the GUI to purchase items of their choosing. Users will also be able to check their shopping carts and remove items from it if they choose to do so.

Functions:

Browse Products:

- The user shall be able to browse database for items that are for sale
- The application shall be able to provide a list of the items available for purchase in a grid-like format and allow the user to select a specific category to view a subset of the database.
- The user will be able to see the each product's name, picture, and price

Display Product Information:

- The user shall be able to click on a product to view more information
- If a user clicks on a product a new page will be loaded that displays additional information about the product. The page will provide the user additional information about the product as well as the option to purchase and add to their wishlist.

Search for Products:

- The user shall be able to search the database for products by typing a product into a search bar.
- The system shall be able to provide the user with any products that match their query. If the system is unable to find suitable products, the user will be notified with a message.

Sort Products by:

- The user shall be able to specify how they would like the products to be sorted.
- The system shall be able to rearrange the products based on the user's selected sorting criteria

Add Products to Shopping Cart:

- The user shall be able to add a specified quantity of a selected product to their shopping cart. The cart will only be accessible by the user that owns it

Edit Cart:

- The user shall be able to edit the quantity of items or remove items from their shopping cart.
- The system shall be able to recalculate the total cost when the cart is edited.

View Cart:

- The user shall be able to view the products and the quantity of each product in their shopping cart, as well as a total cost for all the items.

Edit Personal Information:

- The user shall be able to edit their personal information such as their name, email, shipping address, and payment information.

Create User:

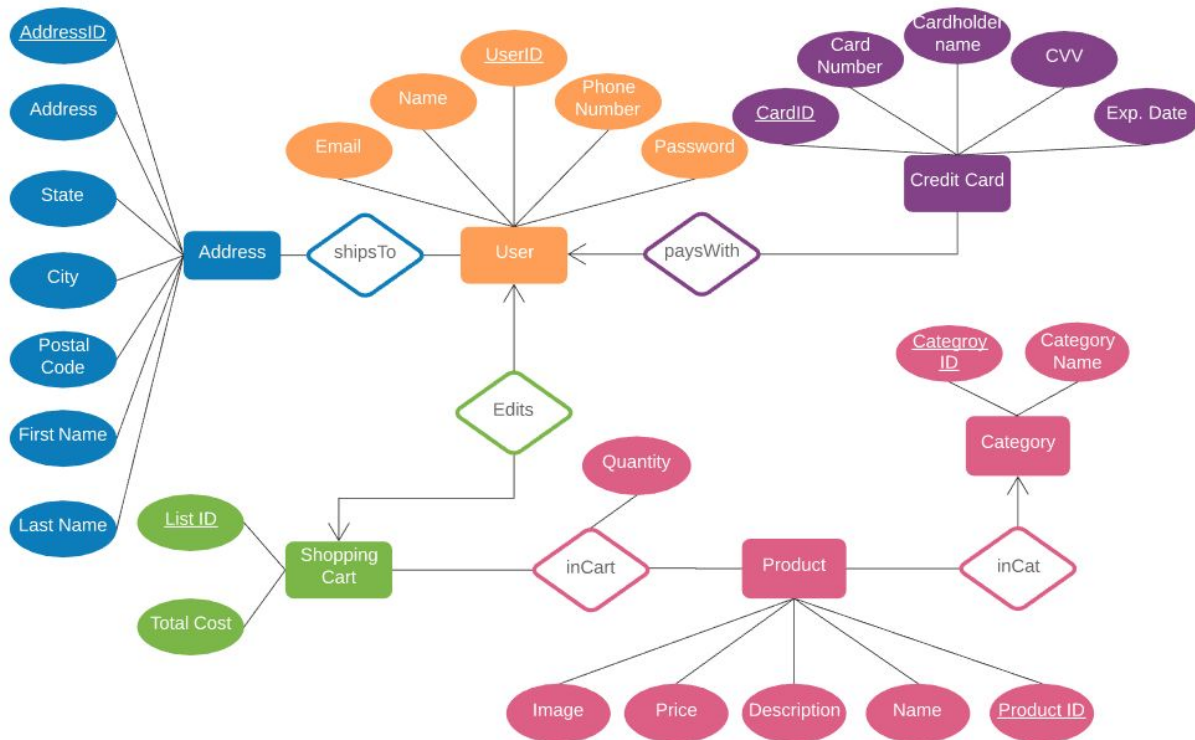
- The user shall be able to create a new account through a provided form. Completion of this form will be required to successfully create the account.
- The system shall be able to prompt users to enter all relevant information and store it for future reference and changes.

Login:

- The user shall be able to login in to their account with the proper credentials
- The system shall be able to reject improper credentials and keep track of the session of the logged in user

Project Details:

ERD:



Entity Sets:

- User - This set represents the registered users of Flipkart. This set store user information.
- Address - This set represents the addresses registered to Flipkart users.
- Shopping Cart - Each user has a shopping cart assigned to them. The shopping cart will store items selected for purchase by the user. The shopping cart can be edited by the user that owns it.
- Credit Card - This set will inherit the primary key from the payment information set. It will store credit card information such as credit card number, CVV, name, and expiration date.
- Product - This set represents all the items that are being sold on FlipKart. It will store information such as a unique product id, product name, description, image, and price.
- Category - This set represents the categories that a product can belong to. It stores a unique category id and the name of the category.

Relationships:

- PaysWith - This relationship is a many to one relationship. It keeps track of the payment information of each user. Each user can have multiple payment methods but each payment method can only be associated with one user.

- Edits - This relationship is one to many. It keeps track of the shopping cart associated with specific users. Each user can have many shopping lists but each shopping list can only be attributed to one user.
- Contains - This relationship is a many to many relationship. It keeps track of the products in a shopping cart. Each shopping cart can have many products and each product can be in many carts
- In - This relationship is a many to one relationship. Each product can only have one category and each category can have multiple products.
- Ships to - This relationship is a many to many relationship. It keeps track of the addresses associated with specific users. Each user can have multiple addresses and the same address can be associated with multiple users.

Entity Schemas:

- User(UserID, Name, Email, Phone Number, Password)
- Address(AddressID, Address, State, City, Zip, First Name, Last Name)
- Carts(CardID, totalCost)
- CreditCard(CardID, CardNum, CardName, CVV, expDate)
- Product(ProductID, Name, Description, Price, Image)
- Category(CategoryID, CategoryName)

Relationship Schemas:

- shipsTo(AddressID, UserID)
- paysWith(UserID, CardID)
- edits(UserID, cardID)
- inCart(CardID, ProductID, quantity)
- inCat(ProductID, CategoryID)

Database Tables:

- User

UserID	Name	Email	PhoneNumber	Password
29711512-efbe-47ab-a253-b940468eead0	Melissa S...	melissa@lol.com	1234567890	12345
541b7d3d-e41f-4727-b81d-2c0242f2a91b	Colin Chow	colin@lol.com	4089587364	12345
5be06677-734e-4ead-b33a-613f23201a94	Leann Zhao	leann@lol.com	8495867384	12345
6cbf8b7c-5289-4366-984b-e437367dc592	Peter Pan	peter@lol.com	6268020549	12345
82bdc38e-d1e5-4633-9a7c-d082139e6bb5	Mike Wu	mike@lol.com	8047385746	12345
8a81cd30-94d6-4f21-ba7c-70ca973ee6c5	Jeff Sayer	jeff@lol.com	7483960594	12345
8f278db7-9230-4383-8e30-0ff8b33b9277	Kevin Lee	kevin@lol.com	6267384759	12345
95380a3f-e6fd-4e64-b92a-b809048d6125	Mathew K...	mathew@lol.com	6375849384	12345
a653dad7-8378-4c10-859c-15715d959ff9	Frank Lin	frank@lol.com	408593827	12345
af2d7de3-95ed-4aef-af05-9a349266b135	Johnathan...	johnathan@lol.com	7463829576	12345
c59362fe-3e11-4b49-8ddb-9404a360c67c	Eunice Sim	eunice@lol.com	4086273849	12345
e79b77c3-b816-4605-86d2-9ab8b05fa43d	Harry Potter	harry@lol.com	80473857482	12345
f16cc27f-9cca-4250-8654-2782ed691798	Darth Vader	darth@lol.com	4089574837	12345
f1cee8f7-f844-4532-aac3-8a18f625f0c3	Danny Song	danny@lol.com	6268020549	12345
f917d020-4c8b-4430-a435-737cc7cb7fc7	John John	john@lol.com	75849374650	12345

- Address

address	state	city	zip	firstName	lastName	addressID
1234 1st St...	CA	Arcadia	95112	Danny	Song	000f6bef-aa0d-4f84-9924-e0426599580b
746 N. 847t...	CA	Arcadia	91006	Leann	Zhao	06d6c223-0b60-41d8-8446-30539490732f
1154 S. 81s...	CA	Arcadia	91006	Johnat...	Smith	107454df-eb86-4f91-96f7-f1d22c9d0843
738 S. 5th St	CA	San ...	95112	Frank	Lin	4b46728b-7d80-47a9-a1e6-a75983dc27c1
7439 Third St.	CA	San ...	95112	Eunice	Sim	5b063cf8-24b4-4027-8a30-9c6d082ba09e
7432 First St	CA	San ...	95112	Colin	Chow	692d7f02-7a56-4345-b275-ae30d9a252e2
8574 S. Fir...	CA	San ...	95112	Mathew	Kessler	706b0294-4149-46c3-972e-9f8c68e4402f
1234 Main St.	CA	Arcadia	91006	Melissa	Song	83352e36-5920-4985-aa1a-65b895f01497
Hogwarts	CA	Som...	91006	Harry	Potter	853cedb1-1b51-4550-88bf-2ba63554acaf
7584 N. 91...	CA	Arcadia	91006	Jeff	Sayer	8975ffc3-20ca-4770-bd7f-69c5cc2481e0
8475 Death...	CA	Space	74934	Darth	Vader	a64a0047-cf78-47f7-a8b6-51dcecd3a100
8373 Seco...	CA	San ...	95112	Mike	Wu	a7f870d2-2594-4510-867d-f3907b608fca
583 N. 15th...	CA	San ...	95112	Kevin	Lee	ab7098c6-b841-465a-9d01-858c327d4899
8409 Fairy ...	CA	Fairies	91006	Peter	Pan	e349dae1-15e3-4336-bc3a-ace2703675a1
1840 Fifth St	CA	San ...	95112	John	John	e657178c-3ee0-4275-8ebd-ef822b4031eb

- Carts

cartID	totalCost
0fc62bb4-213b-419e-b655-7cd8e3ddc8a8	0
20068727-df29-4a37-b5ff-816cfc2ceb9b	0
2c0037af-f8a1-475d-a821-3dbb829bbdf6	0
395ced40-b16a-49c3-8f7d-df641a17cb2b	0
4836113a-d8c5-4d1e-871c-44e789eb0257	0
59daeea0-f1f2-49e3-841d-7d4486b8feb1	0
6bc72b14-f7ab-4ba4-8b00-c4ea92721790	0
6d601156-b8ec-4fdb-ac36-fd50fdc5106d	0
7f81ccb8-fb80-4301-8c57-c21d2a270645	0
838c54bf-b836-4b27-8505-7b0cef9fe6d7	0
8dd73680-3b27-4fee-bc82-15ac7d7a58d8	0
8fac3217-7c92-42c3-86d3-9bfc62e9f930	0
b3124b76-614b-4c5a-bbd0-1efc75fdd24f	0
b56f0ae2-a444-4576-a4fa-d784b508a627	0
e32ec436-3228-40d8-a193-d87493cf3af4	0

- CreditCard

cardNum	cardName	CVV	expDate	cardID
7483954209	Colin Chow	39	09/22	014d24b2-0e79-4532-ab1a-663b7b6773cb
7589406857	Johnathan Smith	758	05/22	208da21e-7e23-4f27-a030-bed42850d119
8573948573	Mike Wu	847	06/21	228ec534-89be-4033-9f4a-860f3cd5be11
7483759405	John John	748	05/22	2804e412-831b-44e6-be9e-60387805d8c8
74859463527	Darth Vader	748	05/22	2f03010d-f559-48f2-a29b-f3e0577b311a
7483726590	Frank Lin	843	05/22	361bd773-2129-4987-a2c6-2699923c17d7
1234567890	Danny Song	738	01/21	366bf41f-4119-438e-a7a8-6372c626d613
7485948372	Harry Potter	999	09/21	7fea1a1d-ea12-455d-a9de-16c22cb59357
7483749506	Eunice Sim	738	05/19	822d807f-972b-431d-ac9c-0c51728b839f
8594736405	Leann Zhao	738	06/21	8f4f0678-eea8-4657-a062-07605133d381
7485726354	Mathew Kessler	748	09/22	c197d2a4-e972-43f4-9785-9753d3da55c7
7483929576	Peter Pan	948	09/22	d55b6dc5-a01a-40e6-8544-4b0089b60697
0987654321	Melissa Song	765	05/22	ef8e0ef1-da15-45bd-b33a-3384bfe35efa
8475920485	Jeff Sayer	174	05/22	f619afbd-1395-42d5-912d-375277a76ba3
7485920396	Kevin Lee	567	06/21	fc7df58c-7b48-496e-aad9-f67805b72cdd

- Product

productID	Name	Description	Price	Image
0c022f3d-1316-11ea-a993-40a3cc604820	Nikon Z6	Mirrorless camera t...	1696.95	z6.jpg
0c0244de-1316-11ea-a993-40a3cc604820	Iron Man 3	Plagued with worry ...	5.99	ironman3.jpg
0c025819-1316-11ea-a993-40a3cc604820	Han Solo	Smuggler. Scoundr...	10.99	hansolo.jpg
0c027521-1316-11ea-a993-40a3cc604820	Free Solo	Professional rock cl...	15.99	freesolo.jpg
0c028e96-1316-11ea-a993-40a3cc604820	Moana	An adventurous tee...	10.99	moana.jpg
42c396ac-131c-11ea-a993-40a3cc604820	Pokemon Sword	Pokémon Sword an...	59.99	sword.jpg
42c3b3d2-131c-11ea-a993-40a3cc604820	Super Smash ...	Super Smash Bros....	59.99	smash.jpg
42c3c5ca-131c-11ea-a993-40a3cc604820	Forza Horizon 4	Forza Horizon 4 is ...	59.99	forza4.jpg
42c3e154-131c-11ea-a993-40a3cc604820	Star Wars Jedi:...	Cinematic, Immersi...	59.99	jedi.jpg
5db0dfb4-12e5-11ea-a993-40a3cc604820	Sony Alpha a7 III	Mirrorless camera t...	1798.99	a7III.jpg
6278d724-12e2-11ea-a993-40a3cc604820	Canon EOS 5...	Comes with spare b...	2499.99	eos5d.jpg
62794da3-12e2-11ea-a993-40a3cc604820	Nikon D850 D...	Comes with with a...	2796.99	D850.jpg
8ffdd141-131c-11ea-a993-40a3cc604820	Cracking the C...	Cracking the Codin...	34.99	ctci.jpg
a788ea25-12d1-11ea-bd11-40a3cc604820	Google Pixel 3	Google flagship with...	549.99	pixel3.jpg
a788f7e0-12d1-11ea-bd11-40a3cc604820	Google Pixel 3a	Budget Google pho...	299.99	pixel3a.jpg
a789ad7f-12d1-11ea-bd11-40a3cc604820	Samsung Gala...	Samsung flagship ...	699.99	galaxys10plus.jpg
a789ee8c-12d1-11ea-bd11-40a3cc604820	iPhone 11 Pro	Expensive	999.99	iphone11pro.png
a789fc7d-12d1-11ea-bd11-40a3cc604820	iPhone 11	Overpriced	699.99	iphone11.png
a78a086d-12d1-11ea-bd11-40a3cc604820	OnePlus 7t	Flagship performan...	599.99	op7t.jpg
a78a0eb9-12d1-11ea-bd11-40a3cc604820	Samsung Gala...	Samsung flagship ...	899.99	galaxy_note.jpg
c07fb85e-131e-11ea-a993-40a3cc604820	Surface Laptop 3	Slim and stylish, wit...	1199.99	surface3.jpg
c07fcc93-131e-11ea-a993-40a3cc604820	Macbook Pro	Apple's most power...	1299.99	macbookpro.jpg

- Category

categoryId	categoryName
1	Phones
2	Cameras
3	Movies
4	Books
5	Videogames
6	Laptops
7	Toys

- shipsTo

userID	addressID
29711512-efbe-47ab-a253-b940468eead0	83352e36-5920-4985-aa1a-65b895f01497
541b7d3d-e41f-4727-b81d-2c0242f2a91b	692d7f02-7a56-4345-b275-ae30d9a252e2
5be06677-734e-4ead-b33a-613f23201a94	06d6c223-0b60-41d8-8446-30539490732f
6cbf8b7c-5289-4366-984b-e437367dc592	e349dae1-15e3-4336-bc3a-ace2703675a1
82bdc38e-d1e5-4633-9a7c-d082139e6bb5	a7f870d2-2594-4510-867d-f3907b608fca
8a81cd30-94d6-4f21-ba7c-70ca973ee6c5	8975ffc3-20ca-4770-bd7f-69c5cc2481e0
8f278db7-9230-4383-8e30-0ff8b33b9277	ab7098c6-b841-465a-9d01-858c327d4899
95380a3f-e6fd-4e64-b92a-b809048d6125	706b0294-4149-46c3-972e-9f8c68e4402f
a653dad7-8378-4c10-859c-15715d959ff9	4b46728b-7d80-47a9-a1e6-a75983dc27c1
af2d7de3-95ed-4aef-af05-9a349266b135	107454df-eb86-4f91-96f7-f1d22c9d0843
c59362fe-3e11-4b49-8ddb-9404a360c67c	5b063cf8-24b4-4027-8a30-9c6d082ba09e
e79b77c3-b816-4605-86d2-9ab8b05fa43d	853cedb1-1b51-4550-88bf-2ba63554acaf
f16cc27f-9cca-4250-8654-2782ed691798	a64a0047-cf78-47f7-a8b6-51dcecd3a100
f1cee8f7-f844-4532-aac3-8a18f625f0c3	000f6bef-aa0d-4f84-9924-e0426599580b
f917d020-4c8b-4430-a435-737cc7cb7fc7	e657178c-3ee0-4275-8ebd-ef822b4031eb

- paysWith

userID	cardID
541b7d3d-e41f-4727-b81d-2c0242f2a91b	014d24b2-0e79-4532-ab1a-663b7b6773cb
af2d7de3-95ed-4aef-af05-9a349266b135	208da21e-7e23-4f27-a030-bed42850d119
82bdc38e-d1e5-4633-9a7c-d082139e6bb5	228ec534-89be-4033-9f4a-860f3cd5be11
f917d020-4c8b-4430-a435-737cc7cb7fc7	2804e412-831b-44e6-be9e-60387805d8c8
f16cc27f-9cca-4250-8654-2782ed691798	2f03010d-f559-48f2-a29b-f3e0577b311a
a653dad7-8378-4c10-859c-15715d959ff9	361bd773-2129-4987-a2c6-2699923c17d7
f1cee8f7-f844-4532-aac3-8a18f625f0c3	366bf41f-4119-438e-a7a8-6372c626d613
e79b77c3-b816-4605-86d2-9ab8b05fa43d	7fea1a1d-ea12-455d-a9de-16c22cb59357
c59362fe-3e11-4b49-8ddb-9404a360c67c	822d807f-972b-431d-ac9c-0c51728b839f
5be06677-734e-4ead-b33a-613f23201a94	8f4f0678-eea8-4657-a062-07605133d381
95380a3f-e6fd-4e64-b92a-b809048d6125	c197d2a4-e972-43f4-9785-9753d3da55c7
6cbf8b7c-5289-4366-984b-e437367dc592	d55b6dc5-a01a-40e6-8544-4b0089b60697
29711512-efbe-47ab-a253-b940468eead0	ef8e0ef1-da15-45bd-b33a-3384bfe35efa
8a81cd30-94d6-4f21-ba7c-70ca973ee6c5	f619afbd-1395-42d5-912d-375277a76ba3
8f278db7-9230-4383-8e30-0ff8b33b9277	fc7df58c-7b48-496e-aad9-f67805b72cdd

- Edits

UserID	cartID
c59362fe-3e11-4b49-8ddb-9404a360c67c	0fc62bb4-213b-419e-b655-7cd8e3ddc8a8
f16cc27f-9cca-4250-8654-2782ed691798	20068727-df29-4a37-b5ff-816cfc2ceb9b
541b7d3d-e41f-4727-b81d-2c0242f2a91b	2c0037af-f8a1-475d-a821-3dbb829bbdf6
f917d020-4c8b-4430-a435-737cc7cb7fc7	395ced40-b16a-49c3-8f7d-df641a17cb2b
af2d7de3-95ed-4aef-af05-9a349266b135	4836113a-d8c5-4d1e-871c-44e789eb0257
5be06677-734e-4ead-b33a-613f23201a94	59daeea0-f1f2-49e3-841d-7d4486b8feb1
f1cee8f7-f844-4532-aac3-8a18f625f0c3	6bc72b14-f7ab-4ba4-8b00-c4ea92721790
82bdc38e-d1e5-4633-9a7c-d082139e6bb5	6d601156-b8ec-4fdb-ac36-fd50fdc5106d
6cbf8b7c-5289-4366-984b-e437367dc592	7f81ccb8-fb80-4301-8c57-c21d2a270645
8f278db7-9230-4383-8e30-0ff8b33b9277	838c54bf-b836-4b27-8505-7b0cef9fe6d7
8a81cd30-94d6-4f21-ba7c-70ca973ee6c5	8dd73680-3b27-4fee-bc82-15ac7d7a58d8
95380a3f-e6fd-4e64-b92a-b809048d6125	8fac3217-7c92-42c3-86d3-9bfc62e9f930
e79b77c3-b816-4605-86d2-9ab8b05fa43d	b3124b76-614b-4c5a-bbd0-1efc75fdd24f
29711512-efbe-47ab-a253-b940468eead0	b56f0ae2-a444-4576-a4fa-d784b508a627
a653dad7-8378-4c10-859c-15715d959ff9	e32ec436-3228-40d8-a193-d87493cf3af4

- inCart

cartID	productID	quantity
0fc62bb4-213b-419e-b655-7cd8e3ddc8a8	0c027521-1316-11ea-a993-40a3cc...	1
0fc62bb4-213b-419e-b655-7cd8e3ddc8a8	a789fc7d-12d1-11ea-bd11-40a3cc...	1
20068727-df29-4a37-b5ff-816cfc2ceb9b	0c027521-1316-11ea-a993-40a3cc...	1
20068727-df29-4a37-b5ff-816cfc2ceb9b	42c3e154-131c-11ea-a993-40a3cc...	1
2c0037af-f8a1-475d-a821-3dbb829bbdf6	0c022f3d-1316-11ea-a993-40a3cc...	1
2c0037af-f8a1-475d-a821-3dbb829bbdf6	6278d724-12e2-11ea-a993-40a3cc...	1
395ced40-b16a-49c3-8f7d-df641a17cb2b	a789ee8c-12d1-11ea-bd11-40a3cc...	15
4836113a-d8c5-4d1e-871c-44e789eb0257	c07fcc93-131e-11ea-a993-40a3cc...	1
59daeea0-f1f2-49e3-841d-7d4486b8feb1	a789ee8c-12d1-11ea-bd11-40a3cc...	1
59daeea0-f1f2-49e3-841d-7d4486b8feb1	a78a086d-12d1-11ea-bd11-40a3cc...	1
6bc72b14-f7ab-4ba4-8b00-c4ea92721790	62794da3-12e2-11ea-a993-40a3cc...	1
6bc72b14-f7ab-4ba4-8b00-c4ea92721790	a788ea25-12d1-11ea-bd11-40a3cc...	2
6bc72b14-f7ab-4ba4-8b00-c4ea92721790	a78a0eb9-12d1-11ea-bd11-40a3cc...	1
6d601156-b8ec-4fdb-ac36-fd50fdc5106d	8ffdd141-131c-11ea-a993-40a3cc6...	1
7f81ccb8-fb80-4301-8c57-c21d2a270645	a789ad7f-12d1-11ea-bd11-40a3cc...	10
838c54bf-b836-4b27-8505-7b0cef9fe6d7	42c396ac-131c-11ea-a993-40a3cc...	1
838c54bf-b836-4b27-8505-7b0cef9fe6d7	42c3c5ca-131c-11ea-a993-40a3cc...	1
8dd73680-3b27-4fee-bc82-15ac7d7a58d8	0c028e96-1316-11ea-a993-40a3cc...	1
8dd73680-3b27-4fee-bc82-15ac7d7a58d8	a789ad7f-12d1-11ea-bd11-40a3cc...	1
8fac3217-7c92-42c3-86d3-9bfc62e9f930	0c027521-1316-11ea-a993-40a3cc...	15
b3124b76-614b-4c5a-bbd0-1efc75fdd24f	0c0244de-1316-11ea-a993-40a3cc...	1
b3124b76-614b-4c5a-bbd0-1efc75fdd24f	a789fc7d-12d1-11ea-bd11-40a3cc...	1
b56f0ae2-a444-4576-a4fa-d784b508a627	0c028e96-1316-11ea-a993-40a3cc...	1
e32ec436-3228-40d8-a193-d87493cf3af4	42c3b3d2-131c-11ea-a993-40a3cc...	1
e32ec436-3228-40d8-a193-d87493cf3af4	a788ea25-12d1-11ea-bd11-40a3cc...	1

- inCat

productID	categoryId
a788ea25-12d1-11ea-bd11-40a3cc604820	1
a788f7e0-12d1-11ea-bd11-40a3cc604820	1
a789ad7f-12d1-11ea-bd11-40a3cc604820	1
a789ee8c-12d1-11ea-bd11-40a3cc604820	1
a789fc7d-12d1-11ea-bd11-40a3cc604820	1
a78a086d-12d1-11ea-bd11-40a3cc604820	1
a78a0eb9-12d1-11ea-bd11-40a3cc604820	1
0c022f3d-1316-11ea-a993-40a3cc604820	2
5db0dfb4-12e5-11ea-a993-40a3cc604820	2
6278d724-12e2-11ea-a993-40a3cc604820	2
62794da3-12e2-11ea-a993-40a3cc604820	2
0c0244de-1316-11ea-a993-40a3cc604820	3
0c025819-1316-11ea-a993-40a3cc604820	3
0c027521-1316-11ea-a993-40a3cc604820	3
0c028e96-1316-11ea-a993-40a3cc604820	3
8ffdd141-131c-11ea-a993-40a3cc604820	4
42c396ac-131c-11ea-a993-40a3cc604820	5
42c3b3d2-131c-11ea-a993-40a3cc604820	5
42c3c5ca-131c-11ea-a993-40a3cc604820	5
42c3e154-131c-11ea-a993-40a3cc604820	5
c07fb85e-131e-11ea-a993-40a3cc604820	6
c07fcc93-131e-11ea-a993-40a3cc604820	6

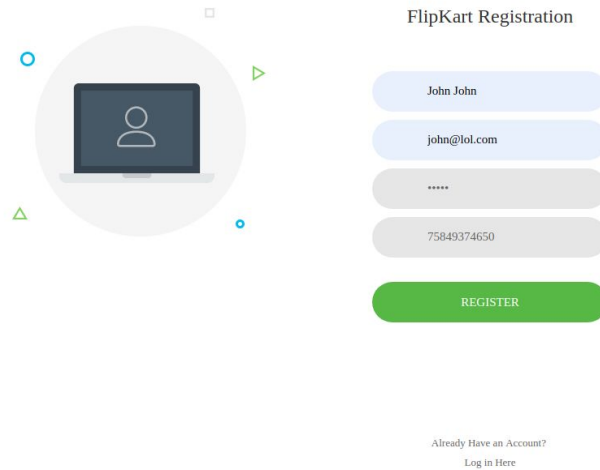
Implementation:

Architecture:

The front end of FlipKart was developed in HTML, CSS, and Javascript. The HTML used the EJS template to allow easier communication with the back end. The MDB and Bootstrap CSS libraries were used to achieve the material design of FlipKart. Javascript was used to validate certain fields such as email on the client side. The web server was developed using Node.JS. The Node server allows the client-end to communicate with the database management system.

Design Implementations:

Create User: Registration UI



The image shows a registration form titled "FlipKart Registration". On the left, there is a circular graphic with a laptop icon and a person silhouette. The form consists of four input fields: a name field with "John John", an email field with "john@lol.com", a password field with "*****", and a phone number field with "75849374650". Below these fields is a green "REGISTER" button. At the bottom, there is a link that says "Already Have an Account? Log in Here".

Database is updated with John John's UID, Name, Email, PhoneNumber, and Password

UserID	Name	Email	PhoneNumber	Password
29711512-efbe-47ab-a253-b940468eead0	Melissa S...	melissa@lol.com	1234567890	12345
541b7d3d-e41f-4727-b81d-2c0242f2a91b	Colin Chow	colin@lol.com	4089587364	12345
5be06677-734e-4ead-b33a-613f23201a94	Leann Zhao	leann@lol.com	8495867384	12345
6cbf8b7c-5289-4366-984b-e437367dc592	Peter Pan	peter@lol.com	6268020549	12345
82bdc38e-d1e5-4633-9a7c-d082139e6bb5	Mike Wu	mike@lol.com	8047385746	12345
8a81cd30-94d6-4f21-ba7c-70ca973ee6c5	Jeff Sayer	jeff@lol.com	7483960594	12345
8f278db7-9230-4383-8e30-0ff8b33b9277	Kevin Lee	kevin@lol.com	6267384759	12345
95380a3f-e6fd-4e64-b92a-b809048d6125	Mathew K...	mathew@lol.com	6375849384	12345
a653dad7-8378-4c10-859c-15715d959ff9	Frank Lin	frank@lol.com	408593827	12345
af2d7de3-95ed-4aef-af05-9a349266b135	Johnathan...	johnathan@lol.com	7463829576	12345
c59362fe-3e11-4b49-8ddb-9404a360c67c	Eunice Sim	eunice@lol.com	4086273849	12345
e79b77c3-b816-4605-86d2-9ab8b05fa43d	Harry Potter	harry@lol.com	80473857482	12345
f16cc27f-9cca-4250-8654-2782ed691798	Darth Vader	darth@lol.com	4089574837	12345
f1cee8f7-f844-4532-aac3-8a18f625f0c3	Danny Song	danny@lol.com	6268020549	12345
f917d020-4c8b-4430-a435-737cc7cb7fc7	John John	john@lol.com	75849374650	12345

Node.JS first queries the user table to see if the email input already is being used. If it is, the user is redirected back to the registration page. If the email has not been used, the user provided information is inserted into the user table. A new cart entity is created in the carts table and the user is associated with that new cart in the edits table.

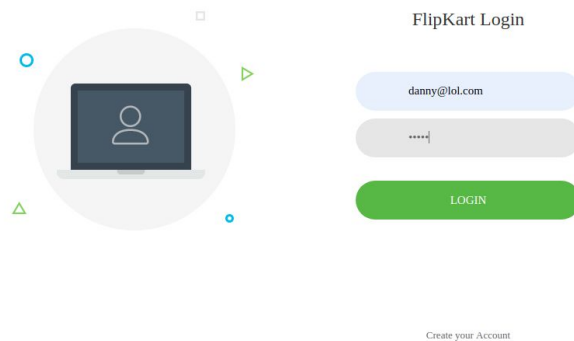

```

let useruuid = uuid.v4();
let addQuery = "INSERT INTO user (UserID, Name, Email, PhoneNumber, Password) VALUES ('"+useruuid+"', '"+name+"', '"+email+"', '"+phoneNumber+"', '"+password+"')";
db.query(addQuery, (err, result) => {
  if (err) throw err;

  let cartUUID = uuid.v4();
  let newCartQ = "INSERT INTO carts (cartID) VALUE ('"+cartUUID+"')";
  db.query(newCartQ, (err, result) => {
    if(err) throw err;
    let cartToUser = "INSERT INTO edits (userID, cartID) VALUE ('"+useruuid+"', '"+cartUUID+"')";
    db.query(cartToUser, (err, result) => {
      if(err) throw err;
      console.log("added new user");
      res.redirect('/');
    });
  });
});
});

```

Login:
Login UI



Node.JS first checks if a user exists with the input email. If a user does exist, the password stored in the database is compared with the password input by the user. The password can be directly compared because the passwords are stored in plaintext as security was not in the scope of this project. Once the password is verified, the session is set with the user's credentials and the user is redirected to the storefront, where they can browse products. If there doesn't exist a user with the inputted email or if the password is incorrect, the user stays on the login page

```

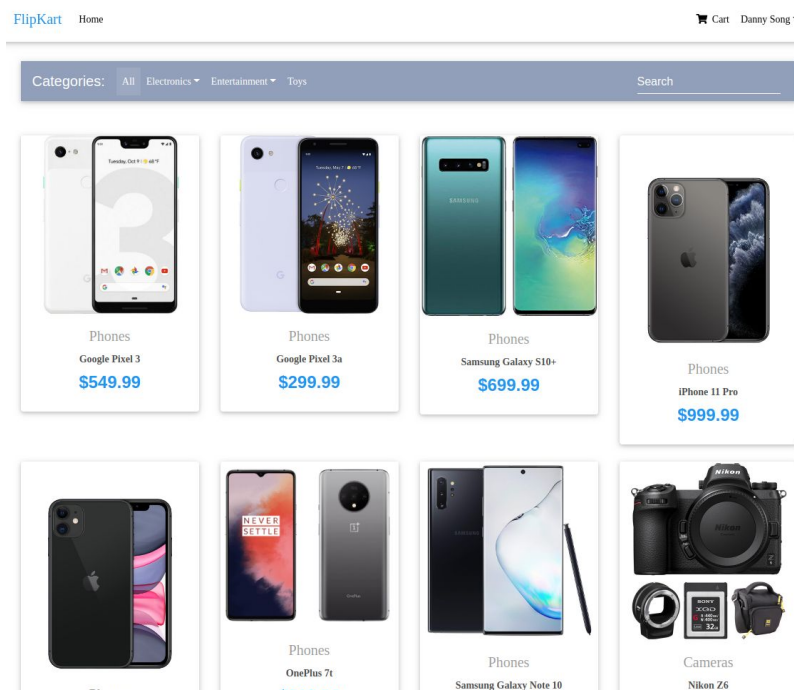
let authCheck = "SELECT * FROM user WHERE Email = '"+email+"'";
db.query(authCheck, function(error, results, fields)
{
  if(results.length)
  {
    if(results[0].Password == password)
    {
      req.session.userId = results[0].UserID;
      req.session.user = results[0];
      res.redirect('/storefront')
      console.log("auth successful:" + results[0].Name);
    }
  }
  else
  {
    console.log("auth unsuccessful");
    res.render('login');
  }
});

```

```
danny@Carbon-X1:~/Documents/CS157a-Team3/FlipKart$ nodemon app.js
[nodemon] 2.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
Server running on port: 5000
Connected to database
auth successful:Danny Song
```

Browse Products:

StoreFront UI

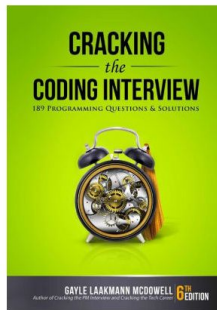


Node JS simply queries the product table for all of the product entities. The result of this query is sent to the storefront.ejs file as products. Each card is individually build for each product using an embedded JS for loop.

```
storeFront: (req, res) => {
  let query = "SELECT * FROM product NATURAL JOIN inCat NATURAL JOIN category";
  db.query(query, (err, result) => {
    if(err) throw err;
    res.render('storefront', {
      products: result});
  })
},
```

Display Product Information:

Product Page UI



\$34.99

Cracking the Coding Interview

Cracking the Coding Interview is here to help you through this process, teaching you what you need to know and enabling you to perform at your very best.

1

ADD TO CART

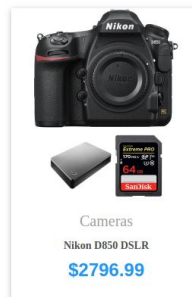
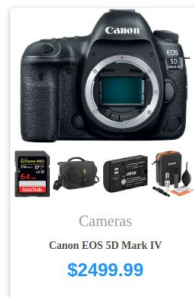
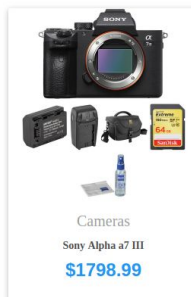
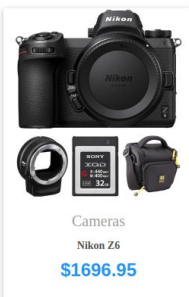
Each product card on the storefront passes its productID as a parameter to NodeJS when the card is clicked. To display the product page, NodeJS queries for the entity in the product table where the productID equals the one passes by the card. The result of this query is sent to productpage.ejs. The product and its attributes are accessed using embedded JS (<%=product.attributeName%>

```
productPage: (req, res) => {
  let id = req.params.id;
  let query = "SELECT * FROM product WHERE productId = '"+id+"'";
  db.query(query, (err, result) =>{
    if(err) throw err;
    res.render('productpage', {
      product: result});
  })
},
```

Sort Products by:
StoreFront UI where cameras is selected

Categories: All Electronics Entertainment Toys

Search



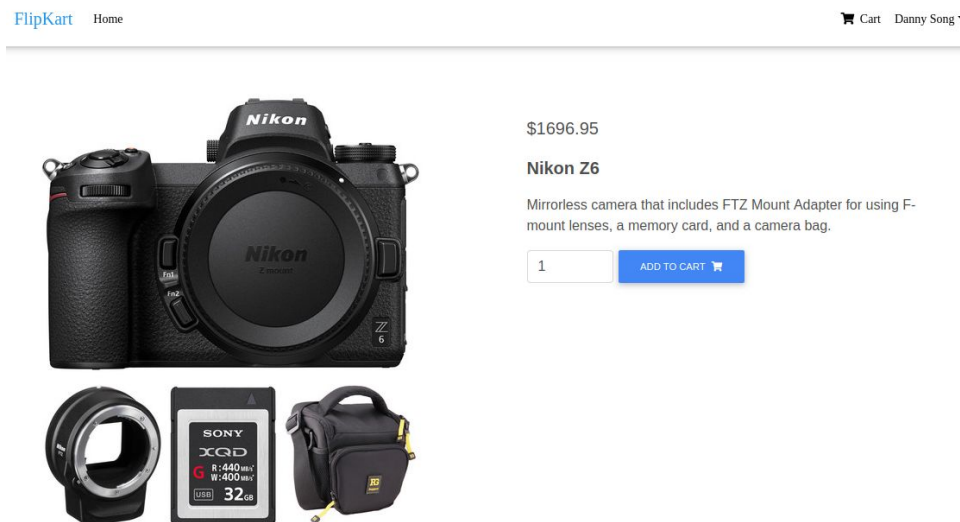
Each button sends the category to NodeJS as a parameter. The product table is joined with the incat table using the product id. Then the resulting table is joined with the category table using the categoryID. This gives us a table with all the product data as well as its category name. Then all products whose category names are the same as the parameter passed by the button are selected. The result is sent to storefront.ejs and displayed in the same way as above.

```
storeFrontCat: (req, res) => {
  let category = req.params.category;

  let query = "SELECT * FROM product NATURAL JOIN incat NATURAL JOIN category WHERE categoryName = '"+category+"'";
  db.query(query, (err, result) => {
    if(err) throw err;
    res.render('storefront', {
      products: result});
  })
},
```

Add Products to Shopping Cart:

Product page with the add to cart button and quantity field



The product is again passed as a parameter by the button to NodeJS while the userID is access from the session we stored at login. NodeJS first queries the edits table to select the cartID that belongs to the user. Once it has the correct cartID, a query is made to insert the cartID, productID, and quantity into the inCart table.

```

addToCart: (req, res) => {
  let productID = req.params.id;
  let qty = req.body.qty;
  console.log(qty);
  let userID = res.locals.user.UserID;
  let cartQuery = "SELECT cartID FROM edits WHERE UserID = '"+userID+"'";
  db.query(cartQuery, (err, result) =>{
    if(err) throw err;
    let cartID = result[0].cartID;
    console.log(cartID);
    let insertQuery = "INSERT INTO inCart (cartID, productID, quantity) VALUES ('"+cartID+", '"+productID+", '"+qty+"'";
    db.query(insertQuery, (err, result) => {
      if (err) throw err;
      console.log("added to cart");
      res.redirect("/product/" + productID)
    })
  })
},




```

cartID	productID	quantity
0fc62bb4-213b-419e-b655-7cd8e3ddc8a8	a789fc7d-12d1-11ea-bd11-40a3cc...	1
20068727-df29-4a37-b5ff-816cfc2ceb9b	0c027521-1316-11ea-a993-40a3cc...	1
20068727-df29-4a37-b5ff-816cfc2ceb9b	42c3e154-131c-11ea-a993-40a3cc...	1
2c0037af-f8a1-475d-a821-3dbb829bbdf6	0c022f3d-1316-11ea-a993-40a3cc...	1
2c0037af-f8a1-475d-a821-3dbb829bbdf6	6278d724-12e2-11ea-a993-40a3cc...	1
395ced40-b16a-49c3-8f7d-df641a17cb2b	a789ee8c-12d1-11ea-bd11-40a3cc...	15
4836113a-d8c5-4d1e-871c-44e789eb0257	c07fcc93-131e-11ea-a993-40a3cc...	1
59daeea0-f1f2-49e3-841d-7d4486b8feb1	a789ee8c-12d1-11ea-bd11-40a3cc...	1
59daeea0-f1f2-49e3-841d-7d4486b8feb1	a78a086d-12d1-11ea-bd11-40a3cc...	1
6bc72b14-f7ab-4ba4-8b00-c4ea92721790	62794da3-12e2-11ea-a993-40a3cc...	1
6bc72b14-f7ab-4ba4-8b00-c4ea92721790	a788ea25-12d1-11ea-bd11-40a3cc...	1

View Cart:

Cart UI

FlipKart Home Cart Danny Song

	Product	Price	QTY	Amount	
 	Nikon D850 DSLR	\$2796.99	1	\$2796.99	+
	Google Pixel 3	\$549.99	2	\$1099.98	+
Total				\$3896.97	CONTINUE TO CHECKOUT

The edits table is joined with the inCart table using cartID. The result is joined with the products table using productID. The resulting table contains each product in each user's cart. From there, the resulting table is queried for the entities where userID equals the userID obtained from the session. Each product and its attributes are accessed and rendered using an embedded JS for loop.



```

cart: (req, res) => {
  let userID = res.locals.user.UserID;
  console.log("user id: " + userID);
  let query = "SELECT * FROM edits NATURAL JOIN inCart NATURAL JOIN product WHERE UserID = '"+userID+"'";

  db.query(query, (err, result) => {
    if(err) throw err;
    res.render('shoppingcart', {
      products: result});
  })
},

```

Edit Cart:

Product	Price	QTY	Amount		
 	Nikon D850 DSLR	\$2796.99	1	\$2796.99	<div>X</div>

Total \$2796.99

CONTINUE TO
CHECKOUT >

```
removeItem: (req, res) => {
  let userID = res.locals.user.UserID;
  let productID = req.body.productID;

  let cartQuery = "SELECT cartID FROM edits WHERE UserID = '"+userID+"'";
  db.query(cartQuery, (err, result) =>{
    if(err) throw err;
    let cartID = result[0].cartID;
    let deleteQuery = "DELETE FROM inCart WHERE cartID = '"+cartID+"' AND productID = '"+productID+"'";
    db.query(deleteQuery, (err, result) =>{
      if (err) throw err;
      res.redirect('/cart');
    })
  })
},
```

cartID	productID	quantity
0fc62bb4-213b-419e-b655-7cd8e3ddc8a8	0c027521-1316-11ea-a993-40a3cc...	1
0fc62bb4-213b-419e-b655-7cd8e3ddc8a8	a789fc7d-12d1-11ea-bd11-40a3cc...	1
20068727-df29-4a37-b5ff-816cfc2ceb9b	0c027521-1316-11ea-a993-40a3cc...	1
20068727-df29-4a37-b5ff-816cfc2ceb9b	42c3e154-131c-11ea-a993-40a3cc...	1
2c0037af-f8a1-475d-a821-3dbb829bbdf6	0c022f3d-1316-11ea-a993-40a3cc...	1
2c0037af-f8a1-475d-a821-3dbb829bbdf6	6278d724-12e2-11ea-a993-40a3cc...	1
395ced40-b16a-49c3-8f7d-df641a17cb2b	a789ee8c-12d1-11ea-bd11-40a3cc...	15
4836113a-d8c5-4d1e-871c-44e789eb0257	c07fcc93-131e-11ea-a993-40a3cc...	1
59daeea0-f1f2-49e3-841d-7d4486b8feb1	a789ee8c-12d1-11ea-bd11-40a3cc...	1
59daeea0-f1f2-49e3-841d-7d4486b8feb1	a78a086d-12d1-11ea-bd11-40a3cc...	1
6bc72b14-f7ab-4ba4-8b00-c4ea92721790	62794da3-12e2-11ea-a993-40a3cc...	1

Edit Personal Information:

Edit Payment UI

Adding a new Card

Flipkart Home

Edit Payment Information

▼ Cart Danny Song •

Danny Song

Full name as displayed on card

01/21

Exp. Date

1234567890

Card Number

CVV

UPDATE
DELETE

Name on card

Full name as displayed on card

Expiration **CVV**

ADD CARD

Flipkart Home

Edit Payment Information

▼ Cart Danny Song •

Kevin Lee

Full name as displayed on card

05/22

Exp. Date

0087654321

Card Number

CVV

UPDATE
DELETE

Danny Song

Full name as displayed on card

12/34

Exp. Date

ADD CARD

Edit Card

Edit Payment Information

<input type="text" value="Darth Vader"/> <small>Full name as displayed on card</small>	<input type="text" value="0987654321"/> <small>Card Number</small>
<input type="text" value="05/22"/> <small>Exp. Date</small>	<input type="text" value="***"/> <small>CVV</small>
<input type="button" value="UPDATE"/>	<input type="button" value="DELETE"/>

<input type="text" value="Danny Song"/> <small>Full name as displayed on card</small>	<input type="text" value="1234567890"/> <small>Card Number</small>
<input type="text" value="01/21"/> <small>Exp. Date</small>	<input type="text" value="***"/> <small>CVV</small>
<input type="button" value="UPDATE"/>	<input type="button" value="DELETE"/>

<input type="text"/> <small>Name on card</small>	<input type="text"/> <small>Credit card number</small>
---	---

Delete Card

Edit Payment Information

<input type="text" value="Danny Song"/> <small>Full name as displayed on card</small>	<input type="text" value="1234567890"/> <small>Card Number</small>
<input type="text" value="01/21"/> <small>Exp. Date</small>	<input type="text" value="***"/> <small>CVV</small>
<input type="button" value="UPDATE"/>	<input type="button" value="DELETE"/>

<input type="text"/> <small>Name on card</small>	<input type="text"/> <small>Credit card number</small>
<input type="text"/> <small>Full name as displayed on card</small>	<input type="text"/> <small>CVV</small>
<input type="text"/> <small>Expiration</small>	<input type="text"/> <small>CVV</small>

Each of the three buttons each send a unique value when pressed, along with the form data, to NodeJS. CardID is stored in a hidden input field in the form. If the delete button is pressed, the entity corresponding to the cardID is deleted first from paysWith then from creditCard. A very similar process happens when an address is deleted from shipping info.

```
if(buttonType == "Delete"){
  let unlinkCard = "DELETE FROM paysWith WHERE cardID='"+cardID+"'";
  db.query(unlinkCard, (err, result) =>{
    if(err)throw err;
  })
  let deleteQuery = "DELETE FROM creditCard WHERE cardID='"+cardID+"'";
  db.query(deleteQuery, (err, result) =>{
    if(err)throw err;
    res.redirect("/mypayment");
  })
}
```

If the user inputs a new card, a new cardID will be generated. This is so insertion into both the creditCard and paysWith tables are possible. A very similar process happens when an address is added to shipping info.

```
else if(buttonType == "newCard"){
  let cardID = uuid.v4();
  let addCard = "INSERT INTO creditCard (cardNum, cardName, CVV, expDate, cardID) VALUES ('"+cardNum+"', '"+cardName+"', '"+cvv+"', '"+expDate+"', '"+cardID+"')";
  db.query(addCard, (err, result) =>{
    if(err) throw err;
  })
  let connectCard = "INSERT INTO paysWith (userID, cardID) VALUES ('"+userID+"', '"+cardID+"')";
  db.query(connectCard, (err, result) =>{
    if(err) throw err;
    res.redirect("/mypayment");
  })
}
```

If the user edits a card, the entity whose cardID equals that set in the hidden field is updated with the values from the form. A very similar process happens when an address is edited in shipping info.

```
else{
  let updateCC = "UPDATE creditCard SET cardName = '"+cardName+"', cardNum = '"+cardNum+"', expDate = '"+expDate+"', CVV = '"+cvv+"' WHERE cardID = '"+cardID+"'";
  db.query(updateCC, (err, result) => {
    if(err) throw err;
    res.redirect("/mypayment")
  })
}
```

Procedure:

1. Run FlipKart.sql script in MySQL to create the database
2. Navigate to FlipKart directory
3. Install dependencies using `npm install`
4. Configure the db const. in app.js to access your MySQL server
5. Start app.js using 'nodemon app.js'
6. Navigate to localhost:5000

Conclusion:

Danny:

I learned quite a few new technologies working on this project. Node.JS, MySQL, and the EJS template were all new to me coming into this project. However, I think the most important thing I gained from this was the opportunity to learn these new concepts and technologies. I think that knowing how to learn new concepts effectively is the most skill that a software engineer can have and I am glad that this gave me the opportunity to do just that. Another thing I learned from this class is the importance of planning code in advance. Looking back at my code now, it looks like kind of a mess and would need quite a bit of refactoring to be readable. I think if I planned it out better, the need to refactor could have been reduced.

As I mentioned before I think the biggest improvement I could make to FlipKart would be the organization of my code. I think a large reason as to why my code is so hard to read is because I was learning as I coded. Thus, the code I wrote in the earlier stages of the project were very rudimentary. Trying to work around this without rewriting everything was a struggle and one I would like to avoid in a future project. I think the UI of FLipKart is also kind of subpar but I really focused on the backend of the project, as that was the emphasis of this assignment. If I had more time, I would have liked to polish the UI even more.