METRO

Online Sneaker E-Commerce Store

Purpose

- To make a E-Commerce store for reselling sneakers.
- Some sneaker start ups only use Instagram to resell sneakers.
- The project aims to grow the stakeholder from a small business to a larger business where they can scale.
- The Project aim is to create an online store that is intuitive, easy to use and provides a personalized shopping experience to the customers.

Similar Projects to Metro

- Prior Store - https://www.priorstoreofficial.com/
- Stock x - https://stockx.com/
- Sneaker Vault NZ - https://sneakervaultnz.com/

Current State of the E-commerce Industry

- Sneaker resellers are generally small businesses that grow fast quickly
- They generally expand into other niches as well such as clothing.
- Most resellers in today's age need a website with a proper backend to expand and handle orders. Social media such as IG and FB marketplace does not provide a scalable solution.

Desired State for Industry

- All small sneaker businesses must be provided with a solution to scale
- Should have more than just a shopify store as an option
- Shopify is expensive, offers limited customisation, has added on costs and does not meet requirements for larger business

Ecommerce Reseller Sneaker Store Value Chain

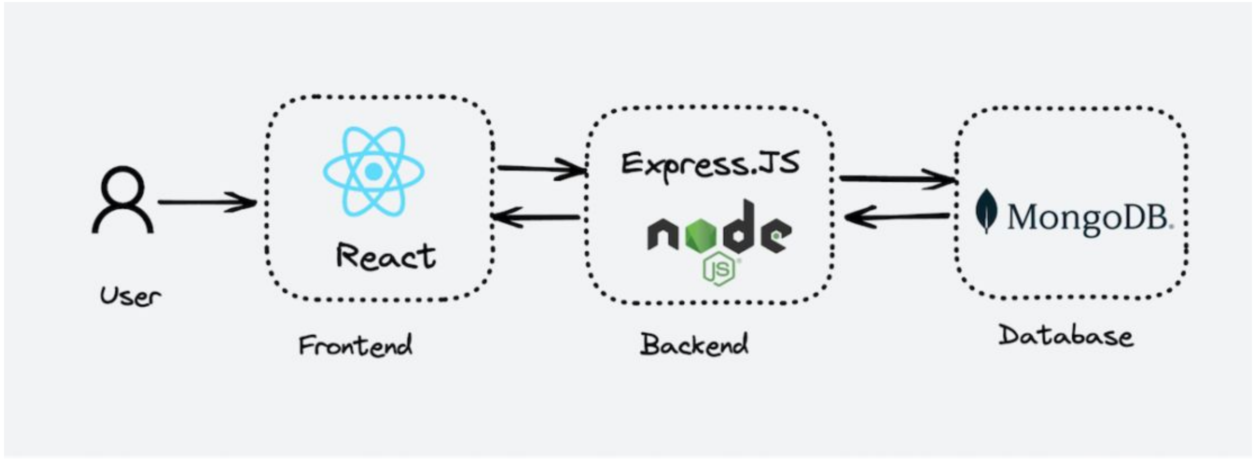
1. Sourcing
2. Authentication & Quality control
3. Storage Inventory Management
4. Platform Management & Online presence
5. Marketing
6. Order Processing, packaging, shipping
7. Customer service
8. Data Analysis

Key Concepts of the sneaker reseller Industry

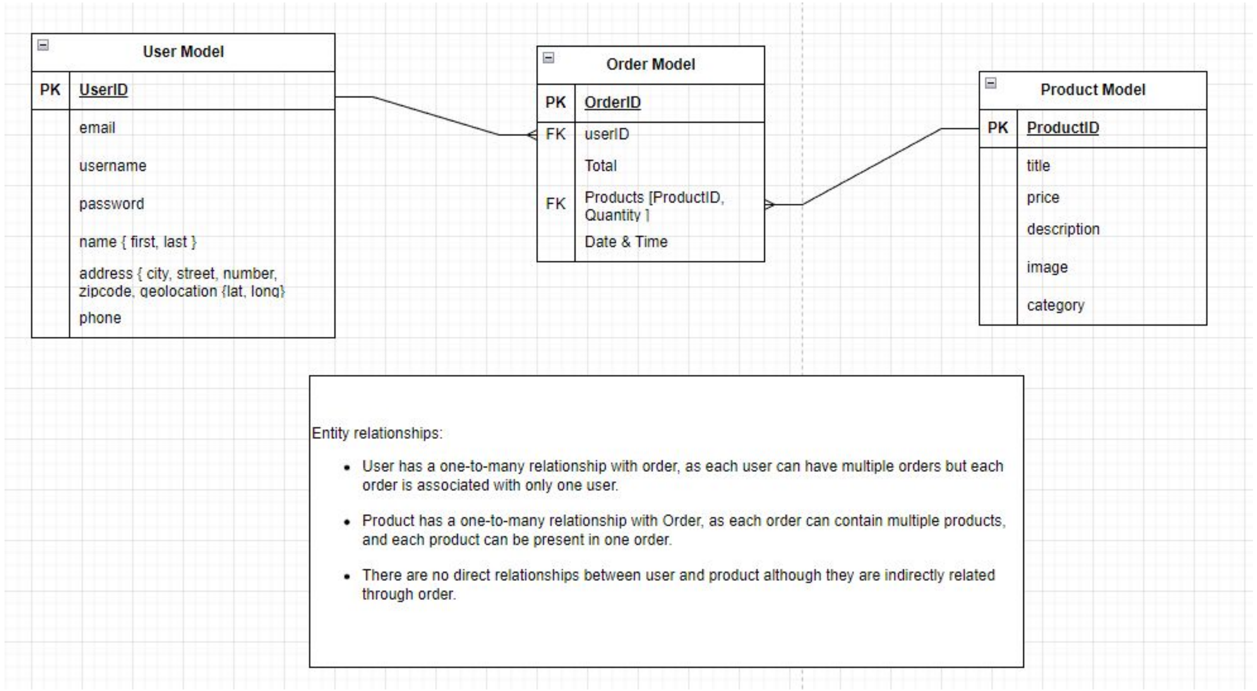
1. Sneaker pricing - based on supply/demand dynamics most cases very scarce which drives high profit margins
2. Connecting Buyers and Sellers using online presence
3. Attracting and retaining customers by an engaging presence
4. Effective inventory management system

System Architecture

Overall the MERN Stack architecture with MongoDB, Express, React and Node.



Backend Architecture

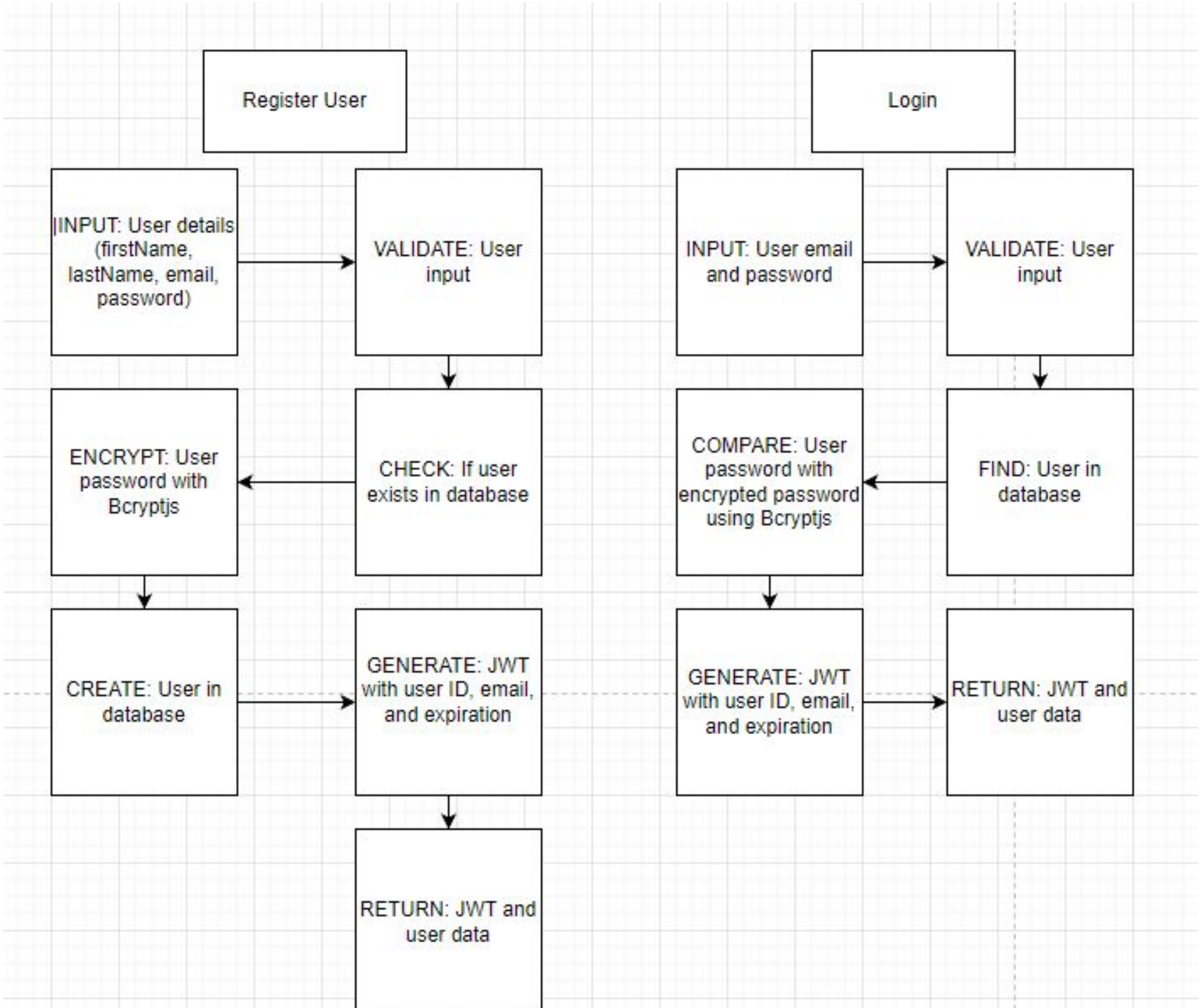


Entity relationships:

- User has a one-to-many relationship with order, as each user can have multiple orders but each order is associated with only one user.
- Product has a one-to-many relationship with Order, as each order can contain multiple products, and each product can be present in one order.
- There are no direct relationships between user and product although they are indirectly related through order.

User Architecture

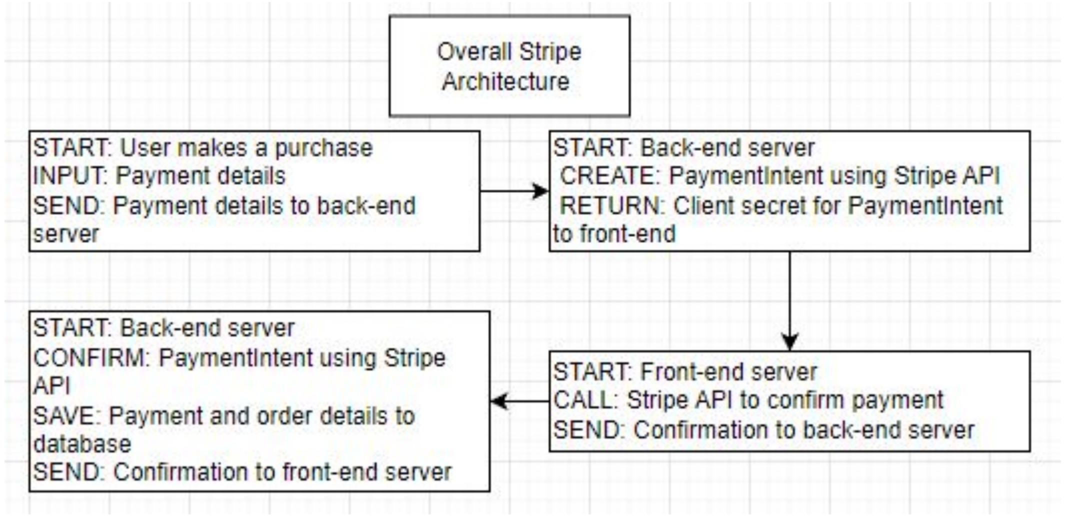
To make the login process secure, the user functions integrate JSON Web Tokens(JWT) for user authentication and authorization


Register User
|INPUT: User details (firstName, lastName, email, password)
VALIDATE: User input
CHECK: If user exists in database
ENCRYPT: User password with Bcryptjs
CREATE: User in database
GENERATE: JWT with user ID, email, and expiration
RETURN: JWT and user data
Login
INPUT: User email and password
VALIDATE: User input
FIND: User in database
COMPARE: User password with encrypted password using Bcryptjs
GENERATE: JWT with user ID, email, and expiration
RETURN: JWT and user data

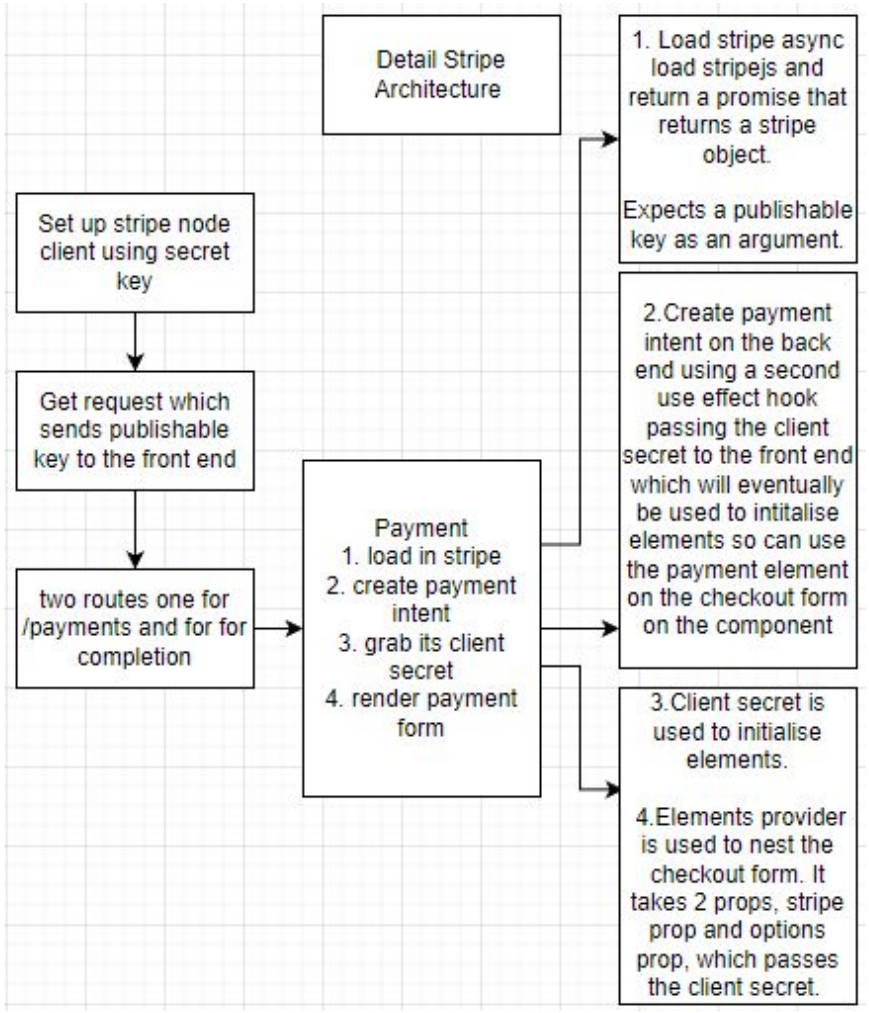
Order Architecture

- To correctly generate an order the order needs to pass an object Id from both the User and the product object ID. This is to provide a security measure that a user that has registered and a product that is in the database can only be created into an order.

Stripe Architecture


Overall Stripe Architecture
START: User makes a purchase
INPUT: Payment details
SEND: Payment details to back-end server
START: Back-end server
CREATE: PaymentIntent using Stripe API
RETURN: Client secret for PaymentIntent to front-end
START: Front-end server
CALL: Stripe API to confirm payment
SEND: Confirmation to back-end server
START: Back-end server
CONFIRM: PaymentIntent using Stripe API
SAVE: Payment and order details to database
SEND: Confirmation to front-end server

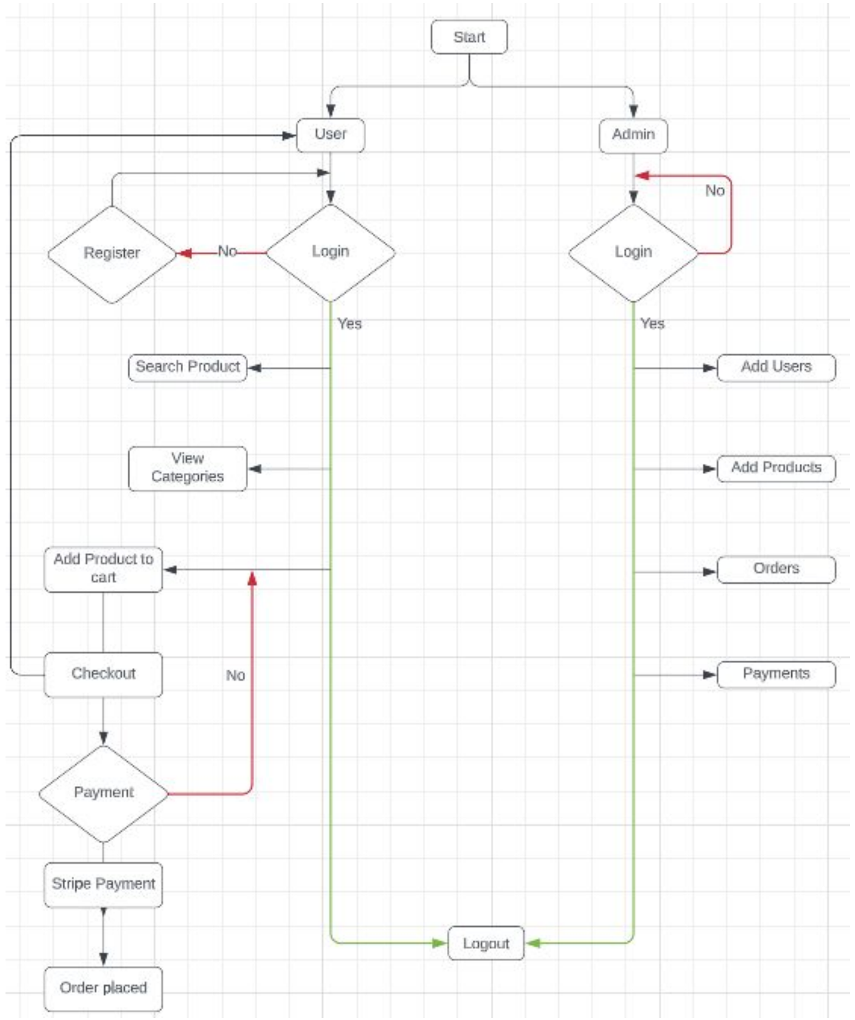
Stripe Architecture More Details


Detail Stripe Architecture
Set up stripe node client using secret key
Get request which sends publishable key to the front end
two routes one for /payments and for for completion
Payment
1. load in stripe
2. create payment intent
3. grab its client secret
4. render payment form
1. Load stripe async load stripejs and return a promise that returns a stripe object.
Expects a publishable key as an argument.
2.Create payment intent on the back end using a second use effect hook passing the client secret to the front end which will eventually be used to intitalise elements so can use the payment element on the checkout form on the component
3.Client secret is used to initialise elements.
4.Elements provider is used to nest the checkout form. It takes 2 props, stripe prop and options prop, which passes the client secret.

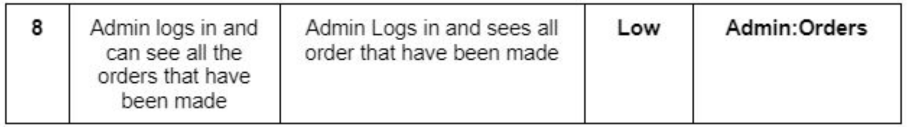
User Flow

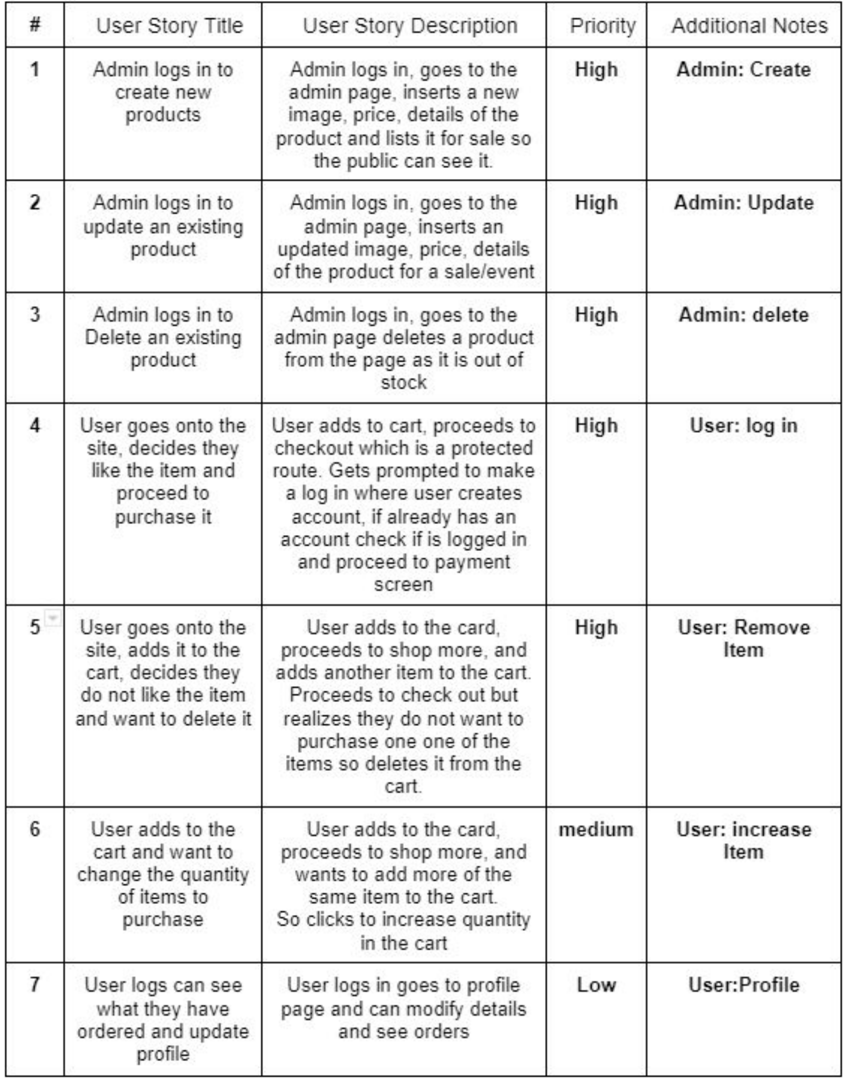
Local storage used to store

- Token
- Cart
- Email



User Stories

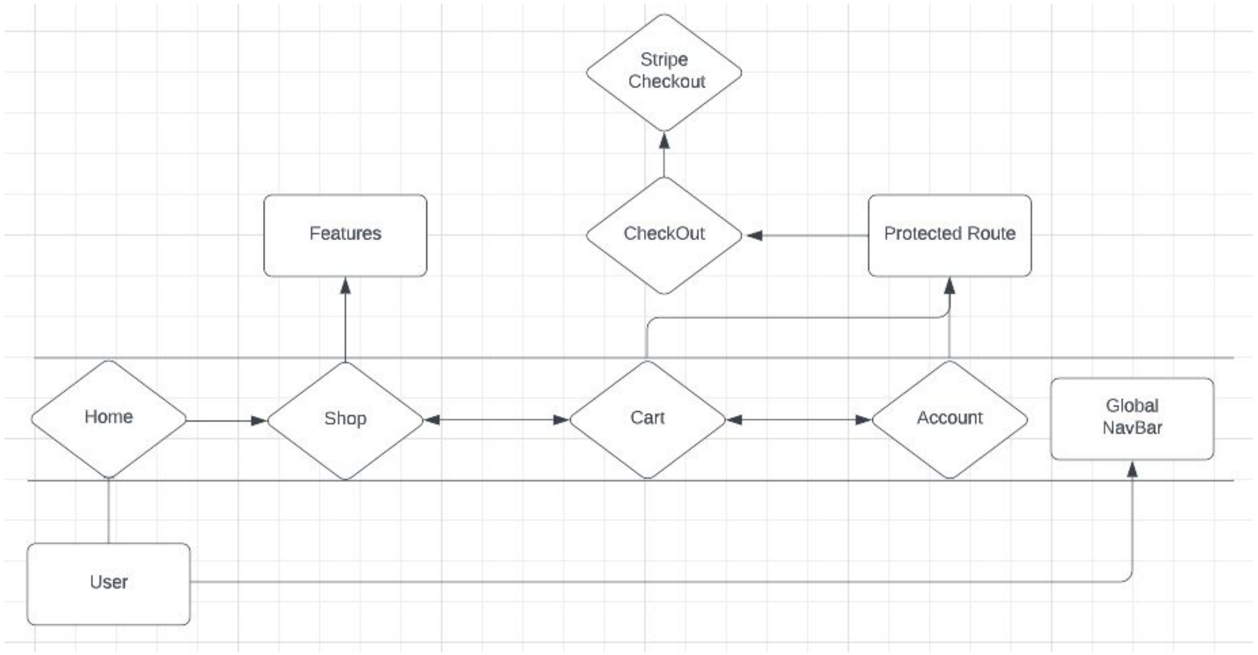




#	User Story Title	User Story Description	Priority	Additional Notes
1	Admin logs in to create new products	Admin logs in, goes to the admin page, inserts a new image, price, details of the product and lists it for sale so the public can see it.	**High**	**Admin: Create**
2	Admin logs in to update an existing product	Admin logs in, goes to the admin page, inserts an updated image, price, details of the product for a sale/event	**High**	**Admin: Update**
3	Admin logs in to Delete an existing product	Admin logs in, goes to the admin page deletes a product from the page as it is out of stock	**High**	**Admin: delete**
4	User goes onto the site, decides they like the item and proceed to purchase it	User adds to cart, proceeds to checkout which is a protected route. Gets prompted to make a log in where user creates account, if already has an account check if is logged in and proceed to payment screen	**High**	**User: log in**
5	User goes onto the site, adds it to the cart, decides they do not like the item and want to delete it	User adds to the card, proceeds to shop more, and adds another item to the cart. Proceeds to check out but realizes they do not want to purchase one one of the items so deletes it from the cart.	**High**	**User: Remove Item**
6	User adds to the cart and want to change the quantity of items to purchase	User adds to the card, proceeds to shop more, and wants to add more of the same item to the cart. So clicks to increase quantity in the cart	**medium**	**User: increase Item**
7	User logs can see what they have ordered and update profile	User logs in goes to profile page and can modify details and see orders	**Low**	**User:Profile**
8	Admin logs in and can see all the orders that have been made	Admin Logs in and sees all order that have been made	**Low**	**Admin:Orders**

User Flow

Users that click on the link are directed to the homepage with global nav.
Users can navigate through home, shop, cart and account.
Shop has features such as category button, search, and add to cart.
Cart has features like remove item, add/subtract quantity and purchase
Account has Login and register.
Check out is a protected route that requires users to have an account.
The user will go through the stripe checkout prompts.



Figma Design

https://www.figma.com/file/SdLloDljdPJjqmJYB9mPnN/Metro?node-id=0-1&t=qFi4cr8cbBGxZpCi-0

Out of Scope

- Using Stripe outside of the Test environment.
- Incorporating google maps api and actual address checking. Currently the user can send to anywhere even a made up address.
- Profile picture images.
- HTTPS integration for security .
- Refunds.
- Ability to become a reseller. Only have admin and user privileges currently.

Non-Functional Requirements

- Login.
- storage of personal details.
- Inactivity timeout after 3 hrs.
- Data encryption.
- Web page loading as fast as possible.
- Images loading as fast as possible.

Expected result + Quality Standards

How many transactions should be enabled at peak time?

- Stripe can handle large amount of volume and to secure checkout by secure means

How easy to use does the software need to be?

- Very user friendly so that users and admin can use it without external help and any user can navigate through most of the site.

How quickly should the application respond to user requests?

- As fast as possible using things like local storage and a reliable backend to fetch data.

How reliable must the application be? (e.g. mean time between failures)

- Very reliable so that if there is any problem it can be solved by refresh or the user can deal with it

Does the software conform to any technical standards to ease maintainability?

- Yes, clean code to conform with the clean code guidelines
- Use Rest API
- Using Stripe to confirm with global security standards
- Ran lots of testing
- Meaningful commented code

Planning

https://trello.com/b/XNh0CnYD/metro

Testing techniques

- End-to-End: User & Admin perspective
- Integration testing: Frontend and backend testing using tools like compass and CRUD operations
- Testing each test case stories

Handling edge cases by checking input validation and error handling, browser development tools.

Future Implementation

Use AWS EC2 instance or use something like vercel.

How well did Metro meet its objectives?

- The project met all objectives and requirements.
- There is one bug in relation to the stripe and order creation.
- If it was to be a fully functional checkout system then a payment success would need to retrieve back a stripe success and then create the order.
- Webhooks and more stripe integration would be needed for a fully functional site deployed using real payments.

References

- Data was originally pulled from https://fakestoreapi.com/ Using a one-off function for users & products which is commented out in userController and productController respectively.The function automatically encrypts the users passwords.

```
cd \Capstone\frontend\vite-project

npm install

npm run dev

cd \Capstone\backend\mongodb-app

npm install

npm run start
```

References

https://github.com/redm3/Capstone

Demo

Questions?