Design Document

Traveling Merchant

Team 10

Victoire Beaufils, Nicholas Dullam, Sabrina Eichenberger, Drew Sittley

Index

Purpose

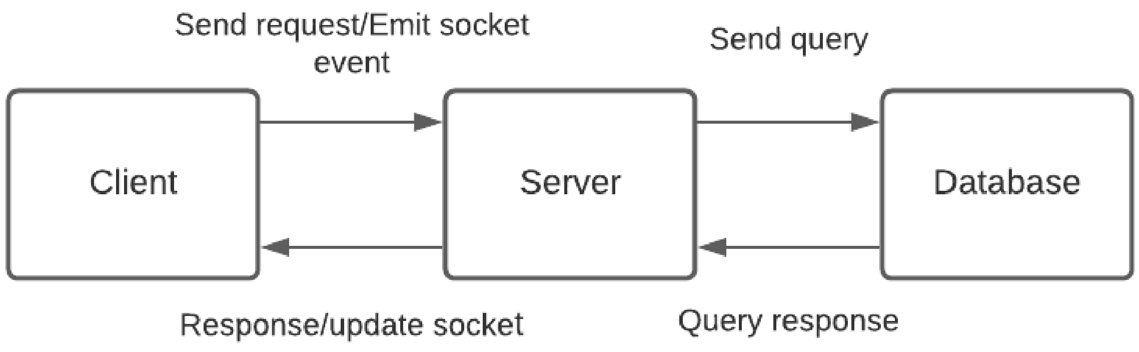
Existing marketplace platforms, such as eBay, don't offer an intuitive user experience or order management for virtual items. Buyers and sellers of virtual goods (i.e. in-game items, accounts, services, and license keys) are often shut down through existing marketplaces inability to verify virtual transactions. Gaming oriented transactions are often left limited to in-game marketplaces, allowing for only account credits or item trading; leaving no intrinsic benefit beyond the scope of games. Targeting gamers for the development of a marketplace for virtual goods and services enables an adapted user experience to compensate for this new generation of ecommerce, while also making 3rd party sales of digital goods more accessible for gamers looking to both buy and sell.

The purpose of this project is to design a web application that allows users to buy and sell game products easily and efficiently. Existing platforms are hard to use and unreliable, so we are striving to amend those issues. We are planning to incorporate an intuitive, transparent, first-party user experience which includes custom orders and outside discovery that leverages a much wider market for sellers than existing counterparts. We also plan to make a front-end discovery page, similar to that of game streaming platforms (i.e. Twitch.tv) that allows for a more gamified and engaging item search. By designing the product page to be more visual, with purchase options more directly accessible also simplifies the checkout experience for buyers. Lastly we plan to incorporate a more scalable database schema that not only makes our site more legitimate for the end-user, but allows for later expansion of game categories and a generally larger target market.

Design Outline

High-Level Overview

This project is a web application that allows users to sell and buy game services and items. This service will use the client-server architecture with a database to respond to both https requests and socket events. The server will be able to handle a large number of events using node.js with express.js as the backend.



1. Client
    a. Client provides a user interface for users
    b. Client sends https requests to show various pages in the server
    c. Client makes socket events to allow real-time messaging and transactions
2. Server
    a. Server receives and handles https requests
    b. Server makes updates as socket events occur
    c. Server receives information and makes updates to the database accordingly, and retrieves data as needed
    d. Server responds to requests accordingly
3. Database
    a. Relational database to store user's information, item information, etc

b. Database responds to queries and stores, retrieves, and updates information

Design Issues

Functional issues

(website layout?, seller oriented/buyer oriented/both, messaging system capabilities?, product mix/inventory?, refund/exchange policies, transaction costs)
Notes: start with listings, progress to requests

1. What marketplace structure should we follow

- Option 1: Seller-oriented
- Option 2: Buyer-oriented
- Option 3: Both buyer and seller-oriented

Choice: Option 3 is our choice for the application
Justification: When it comes to developing an interactive web-app, it is necessary to consider both audiences; especially in this given case of a virtual marketplace. Allowing the buyer to request unlisted products, and sellers to offer products for ease of use on the buyer allows for a much more interactive, and rewarding experience for both parties. Existing marketplace/freelance platforms including Upwork and Fiverr take a very similar approach. Fiverr starting with a seller-oriented platform and moving towards a mix, and Upwork starting as buyer-oriented and also moving towards including seller-oriented features. Choosing to encompass both buyer and seller oriented markets also prepares us for the development of a scalable platform that would be capable of handling the task later down the road.

2. What method of payment processing should we follow

- Option 1: Destination Charges
- Option 2: Direct Charges

Choice: Option 1 is our choice for the application
Justification: When considering destination charges vs. direct charges, we chose to include destination charges for seller interactions. Destination charges run through our platform's account first, following through to the seller afterwards. Direct charges on the other hand, go directly to the seller, giving the platform commissions after the fact. While destination charges would limit the payout period to the seller, this would enable us to more effectively offer refunds, dispute periods, and receive platform commissions while taking the charges from Stripe's processing. Not only would this

make the buyer's experience more flexible and trusting, it would make our role as the platform much more financially secure.

3. What form of commission should we charge for transactions
- Option 1: Flat-rate
- Option 2: Percentage
- Option 3: Variable Percentage

Choice: Option 3 is our choice for the application
<u>Justification</u>: Given our stance as a gaming-oriented market and platform, we opted to include a variable percentage on our commissions. By variable, we are alluding towards a decrease in commissions depending on a set of factors for the seller (i.e. total sales volume, ratings, age of account). This system is not only scalable when held against flat-rate commissions, but provides a more gamified user experience for the seller that rewards them for their continued use of the platform for sales; something that would be lacking in both options 1 and 2.

4. What capabilities should or messaging system have
- Option 1: Read Receipts
- Option 2: Read Receipts and attachments
- Option 3: Read Receipts, attachments, and forms

Choice: Option 2 is our choice for the application
<u>Justification</u>: When we were considering our messaging system, one core idea that could hurt the user experience would be overuse. Pushing buyers to reach out through messages could prove both annoying, and unreliable for either end of the transaction. Rather than including all of the features in option 3, we opted to make the messaging system both competitive with 3rd party applications through both read receipts and attachments, while also not forcing interactions to be made through it. The inclusion of the forms, while it could make gathering information by the seller much more streamlined, we instead opted to include the form feature in the checkout for products. This requirement makes messaging less necessary, and reduces the ideal time for delivery through our application as the seller will have all requested information at the time of checkout.

Non-functional issues

1. What frontend framework should we use?
- Option 1: Angular

- Option 2: React
- Option 3: Vue.js

Choice: Option 2 is our choice for the application
<u>Justification</u>: In terms of developer usage, React seems to be a framework with one of the fastest growing rate, along with Vue.js (Github star history https://www.codeinwp.com/blog/angular-vs-vue-vs-react/, 2018). In 2021, React was the second most preferred web framework, right below jQuery.js (https://www.resourcifi.com/blog/react-vs-angular-vs-vue/). Its high compatibility with Node.js, Express and MongoDB makes it there exists a lot of resources available online.

2. What tool should we use to build our server?
- Option 1: Node.js
- Option 2: Apache

Choice: Option 1 is our choice for the application
<u>Justification:</u> Node.js has great compatibility with Express and MongoDB. There are a lot of resources online that pertain to the use of this technology stack, especially with React. Apache runs on Linux, and hosting the server on a cloud application platform like Heroku would require a third-party buildpack, which would complicate the process. We do not wish to increase development time by choosing a framework that we are not familiar with as our focus is primarily on creating a user interface that is more intuitive and easy to use, which will be primarily reflected by the frontend development.

3. What DBMS should we use?
- Option 1: MYSQL
- Option 2: MongoDB
- Option 3: Firestore

Choice: Option 2 is our choice for the application.
<u>Justification:</u> We wanted a database with a document model with a JSON like structure instead of a relational database system. This is because some of our team members aren't familiar with database technologies, and the JSON syntax is easy to understand.

4. Which cloud platform should we use to host our server?
- Option 1: AWS
- Option 2: GCP
- Option 3: Heroku

Choice: We chose option 3.

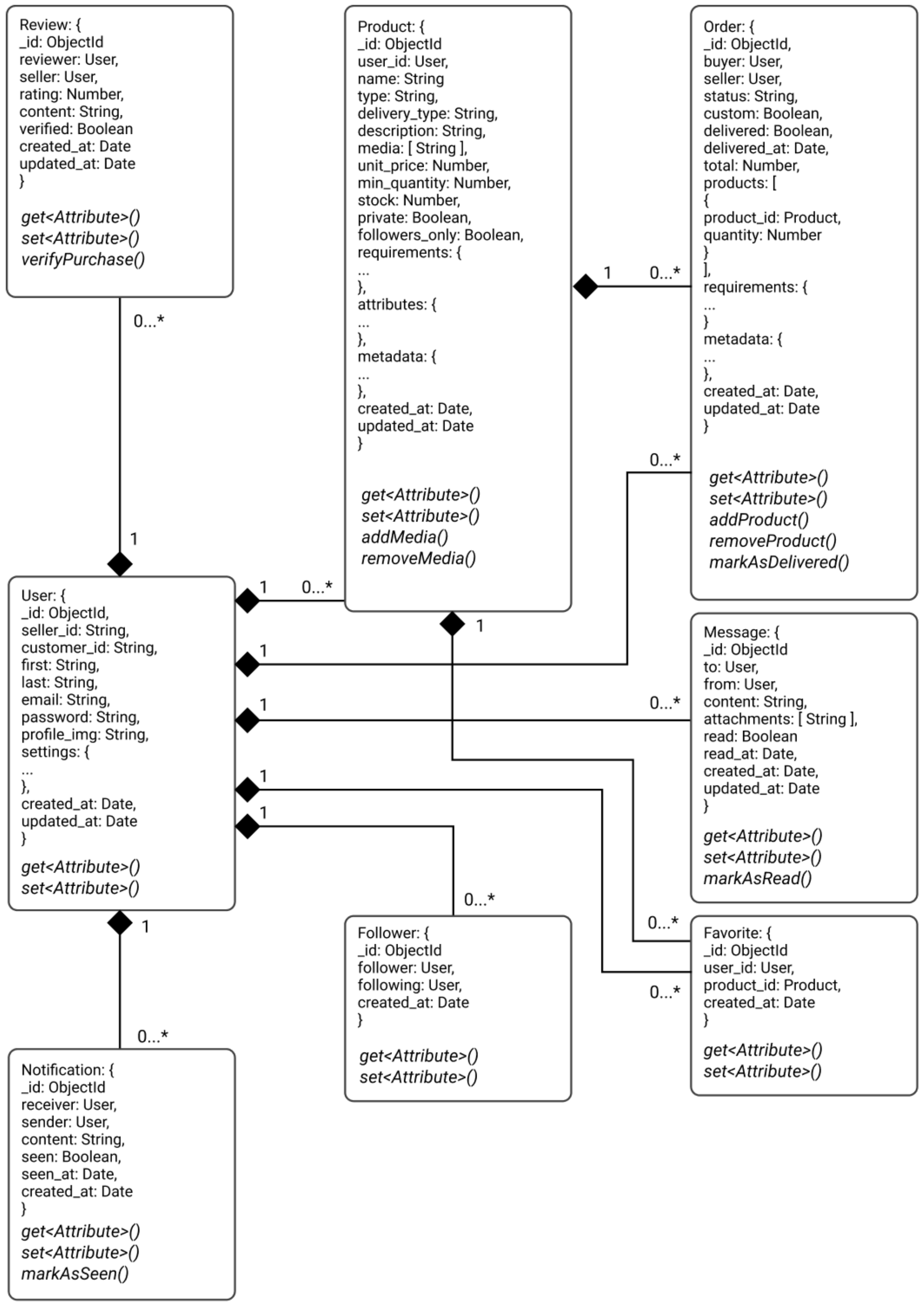
Justification: Heroku provides and renews SSL certificates automatically for us to use HTTPS, and can facilitate the use of continuous delivery with its fast and easy integration with Github.

5. What commercial API should we use to handle payment?
- Option 1: Stripe
- Option 2: Braintree (Paypal)

Choice: We chose option 1.
Justification: While both platforms are fairly similar in most aspects, but the tool provides more customization and better integration with the system. Instead of working around the API, just like with Braintree, developers can choose how to integrate Stripe and its different functionalities. Other game asset products don't provide smooth integration between their website and payment processing system, and it is something that we want to focus on in the development stage, and present as a selling point to our customer.

Design Details

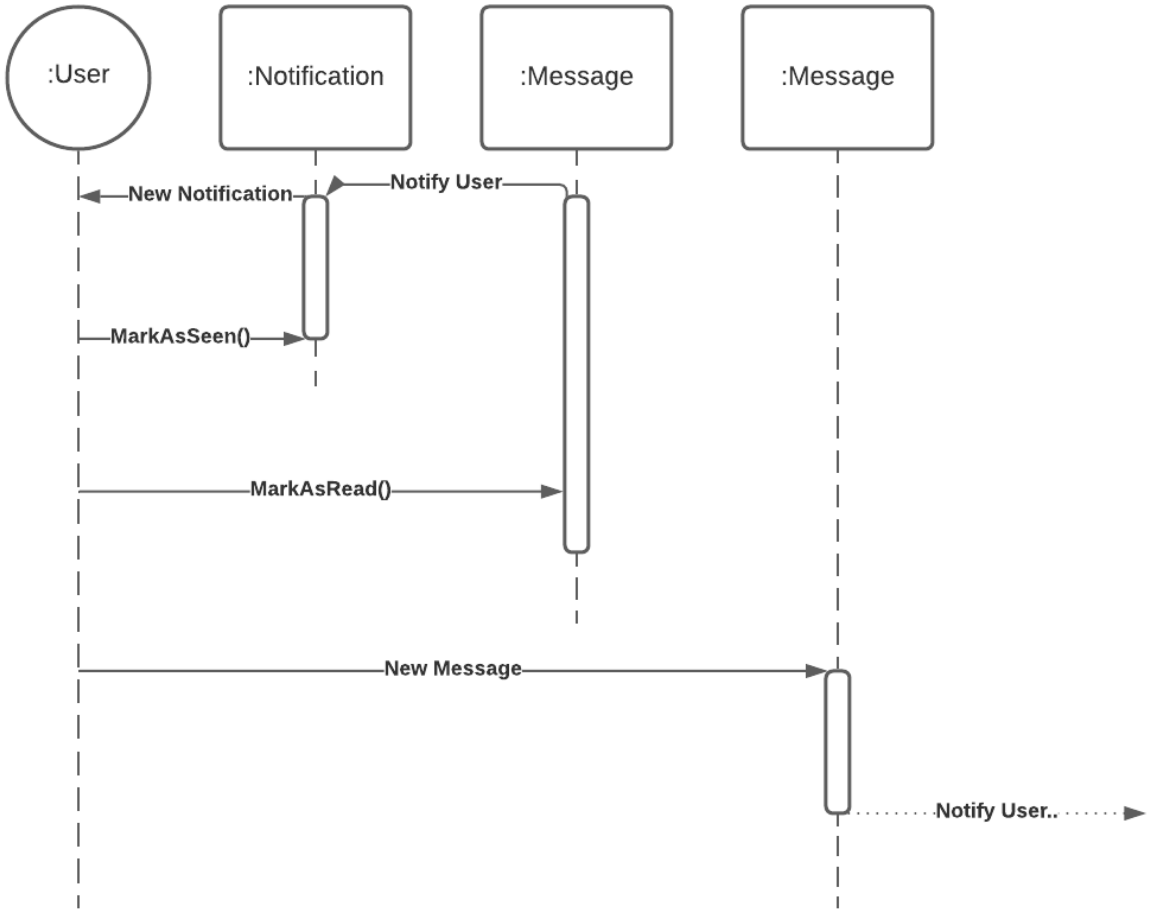
Class Level Design



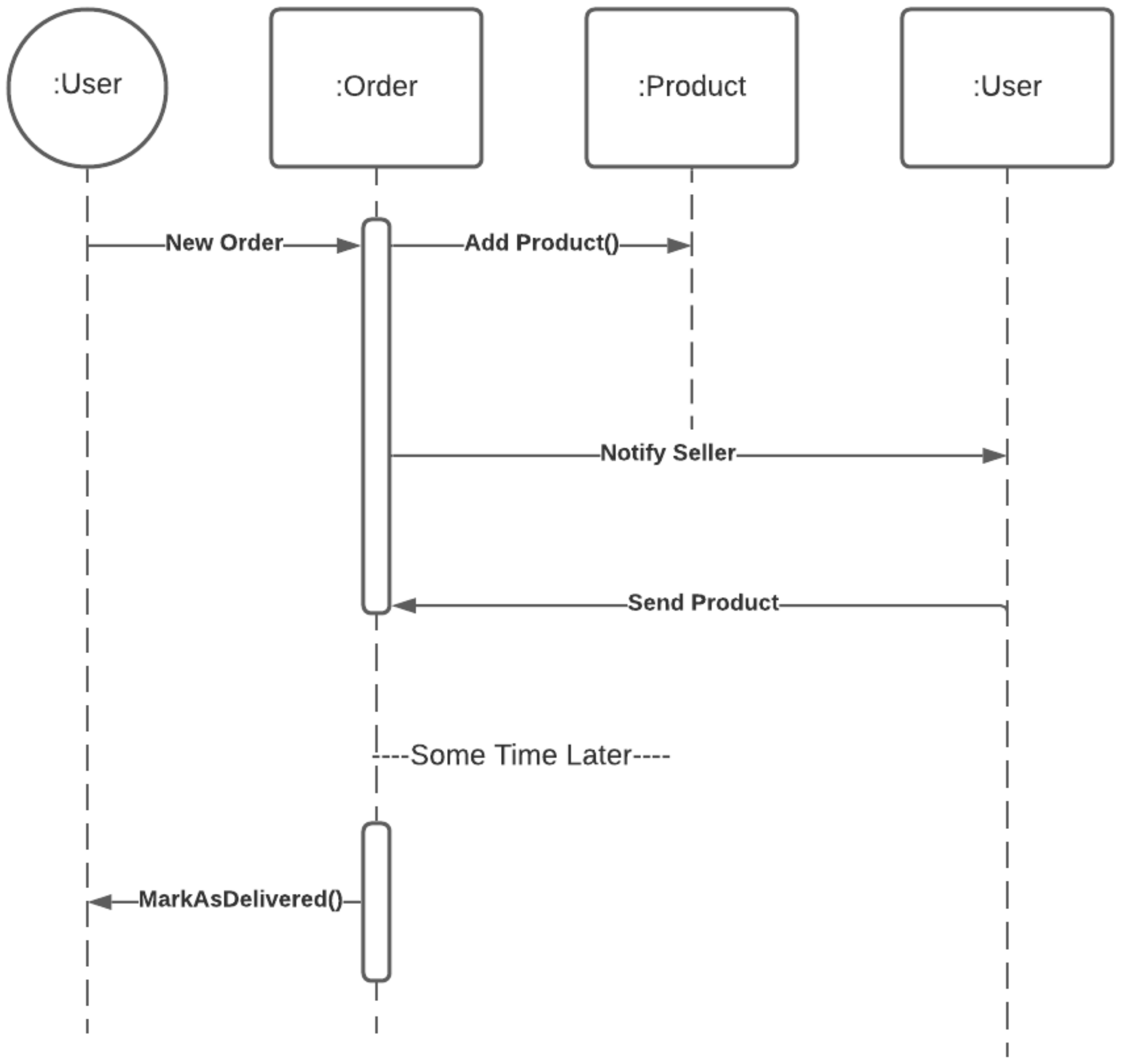
Users will be able to create reviews for sellers, which can be verified through a purchase history with the seller. Users can receive notifications, which includes unseen content, marking them as seen upon opening the notification. Users can also follow other registered sellers, listing said users products on their feed. Users can create favorites for target products, allowing for quicker and easier access. Users can also take part in direct messaging, sending content and attachments to a target seller or buyer with read receipts. Users, when registered as a seller, can create products with the ability to add/remove their target media, and edit attributes including form requirements, and extended information for the product pages. Users can create orders, both from the buyers side and the seller side (custom orders), referring to target products and leading towards purchase. Products can be added or removed from an order relating to a specific seller.

Sequence Diagrams

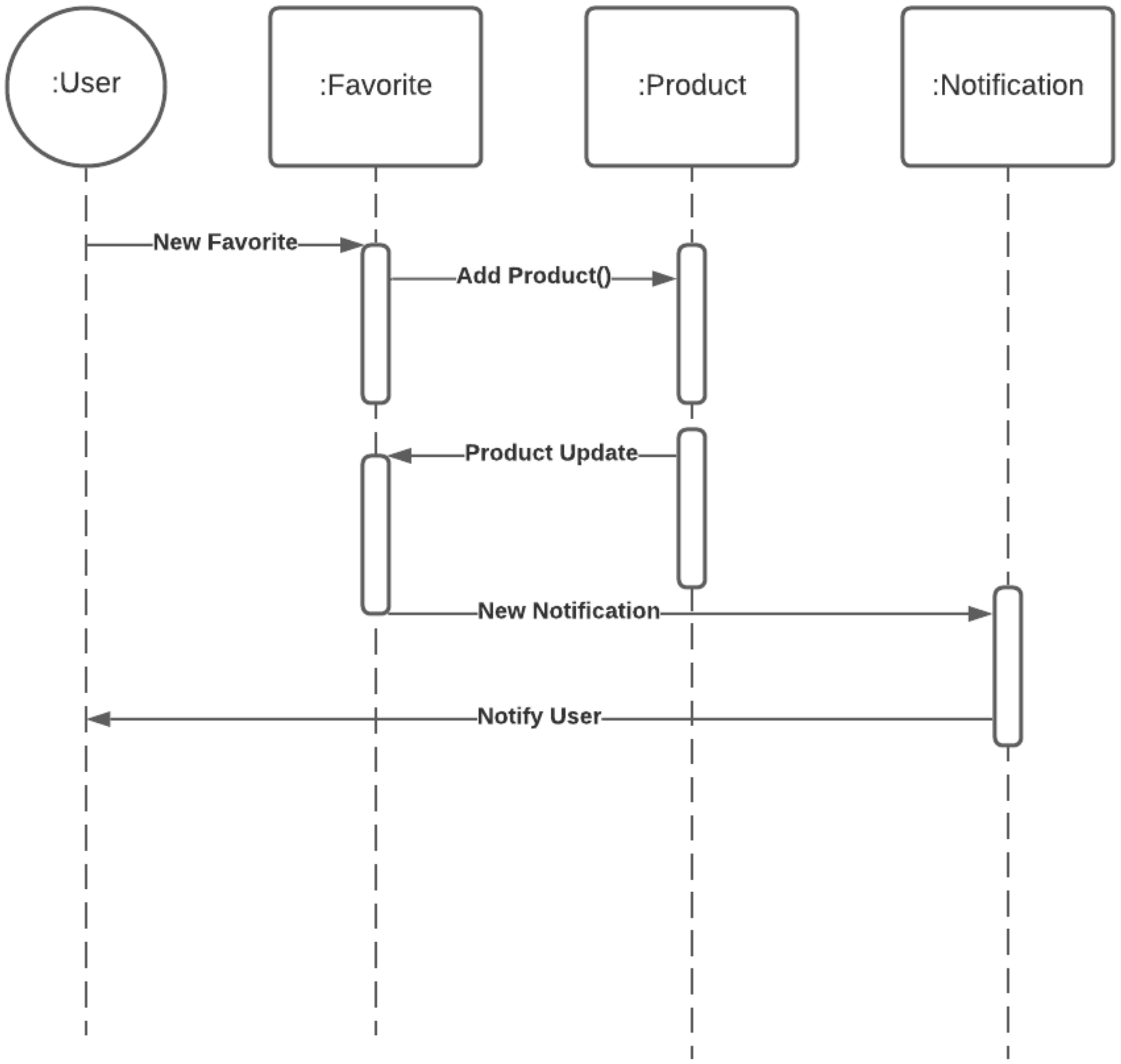
Sample Event 1 - Responding to notification



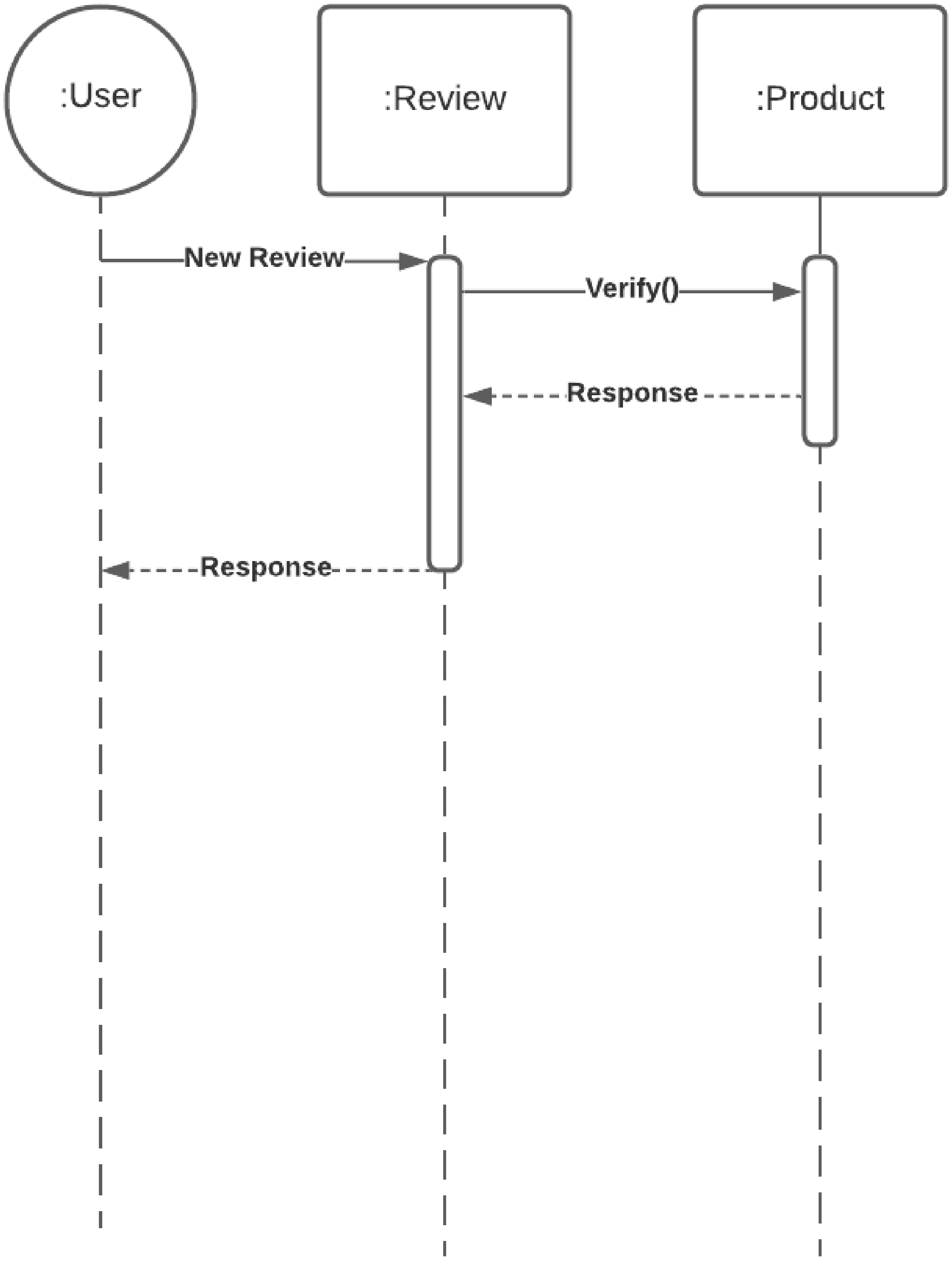
Sample Event 2 - Ordering products



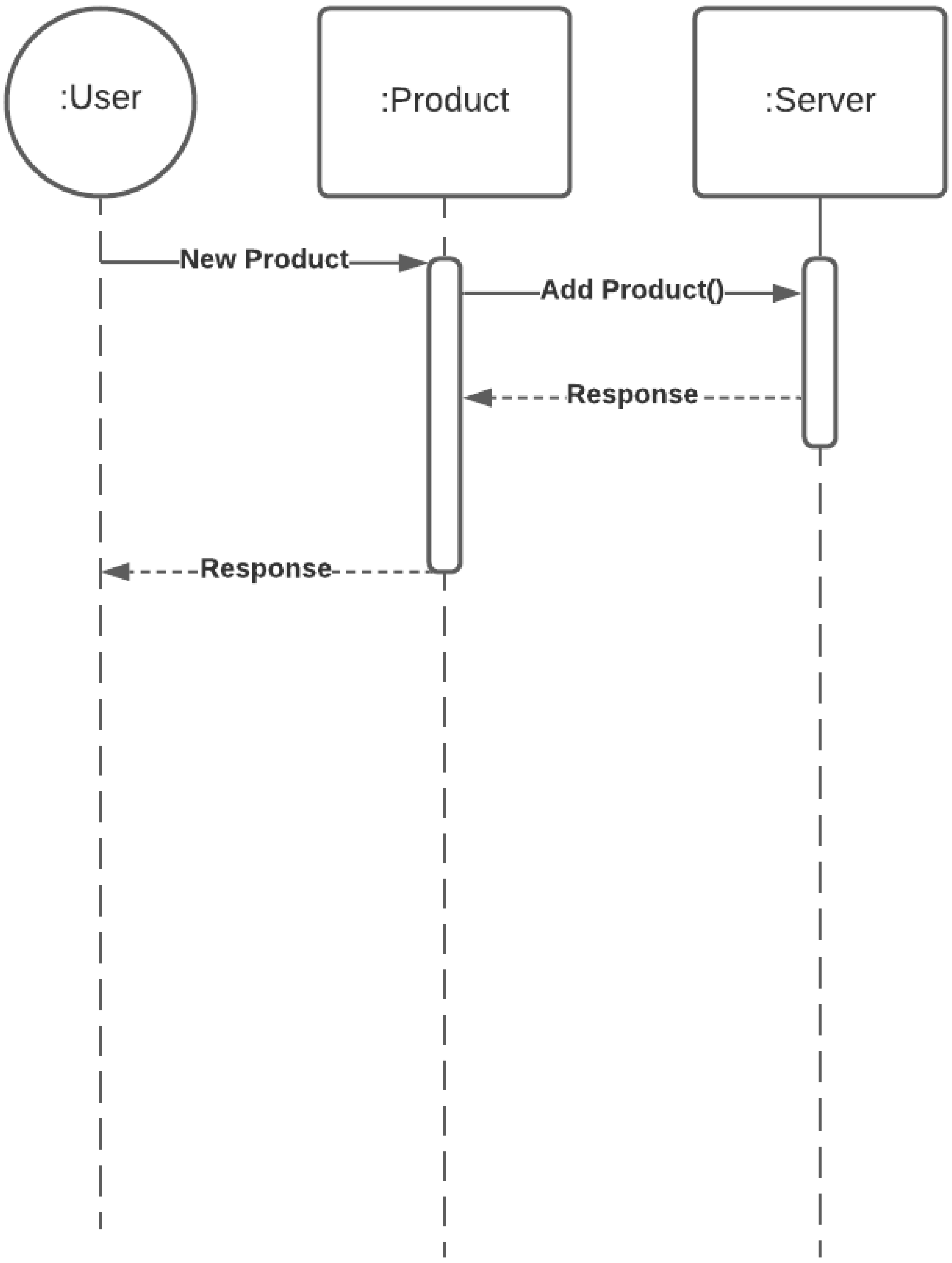
Sample Event 3 - Favorite a product



Sample Event 4 - Review a product

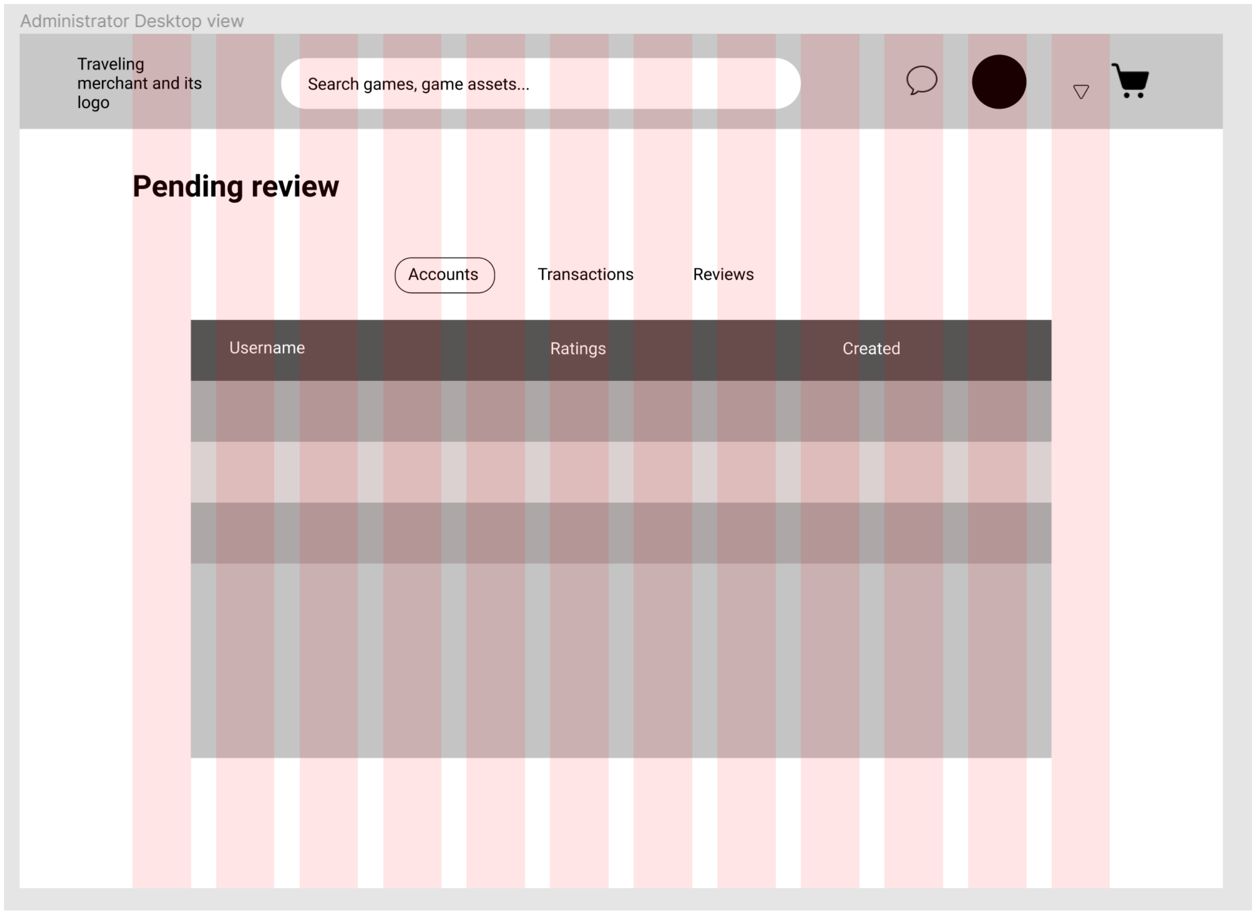


Sample Event 5 - Add a product

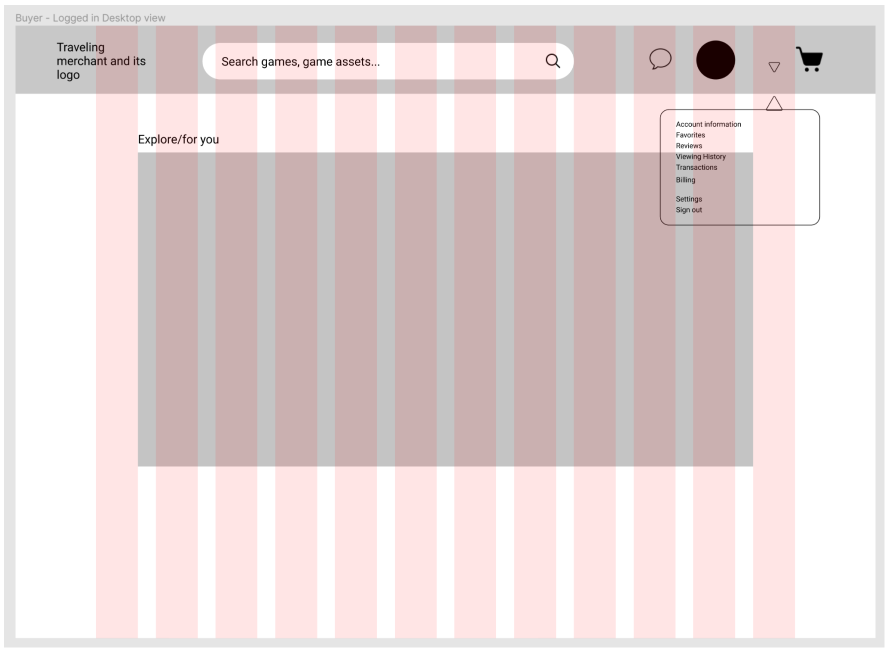


UI Mockup

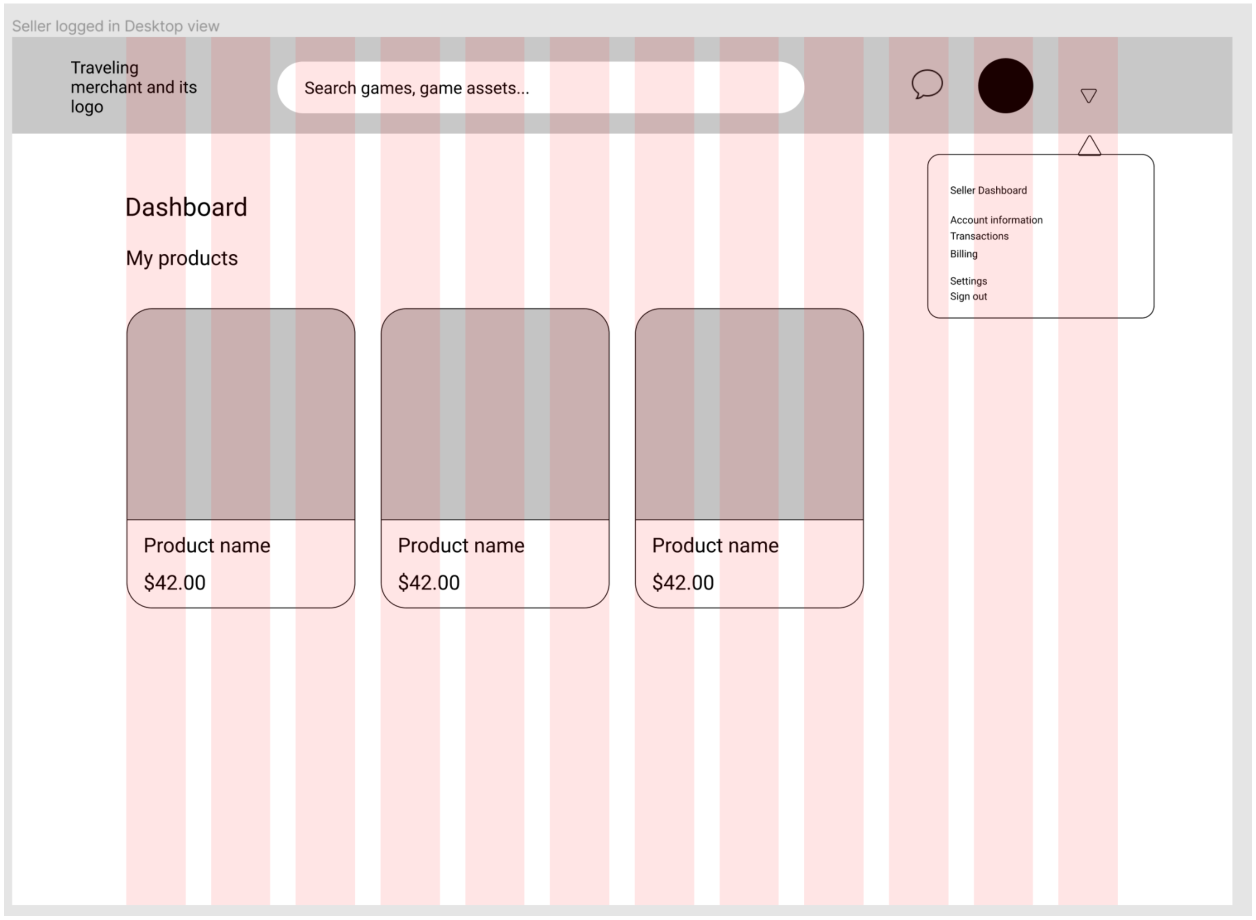
Administrator view


Administrator Desktop view
Traveling merchant and its logo
Search games, game assets...
Pending review
Accounts
Transactions
Reviews
Username
Ratings
Created

Buyer view



Seller view


Seller logged in Desktop view
Traveling merchant and its logo
Search games, game assets...
Dashboard
My products
Product name
$42.00
Product name
$42.00
Product name
$42.00
Seller Dashboard
Account information
Transactions
Billing
Settings
Sign out