**The Game Shop (Website)**

CS 3773-001 Software Engineering

The University of Texas at San Antonio

December 1, 2022

Hezron Perez, Martin Sanabia, Marco Maldonado, Derek Gaura



**Project Description**

We are a group of four Computer Science students with a concentration in software engineering and have been tasked with using our preferred web technologies to create a live website. This website utilizes a database and allows end users to create and edit purchases on the items listed on the website. In our case, the items listed will be games. Now completed, we have experience with full stack development, designing databases and an intuitive user interface (UI). During the project, we utilized software engineering methods like scrum and agile.

**Vision statement**

For gamers of all levels, as well as the friends and family of gamers. The Game Shop is a retail platform that allows anyone from across the world to purchase any game quickly and easily. Unlike our competitors, which mine user data and don’t care about having unique relationships with their customers, such as Google Stadia, the focus of the Game Shop is not only to distribute games effortlessly, but also to give each the chance to have their voice be heard. Our product enables gamers to buy games they couldn’t buy elsewhere.

**Technology Description and Selection**

Initially, we chose the well known technology stack known as the PERN stack combined with Knex JS:

* **PostgreSQL** as the locally hosted database (**prototype/development**), which uses the SQL language to make queries to the database. We went with SQL as our underlying database language for simplicity.
* **Express JS** to run the server locally (ie serve the components from a socket, **prototype/development**)
* **React JS** to create and implement the frontend html components (**prototype/development & production**)
* **Node JS** as the server environment (**prototype/development & production**)
* **Knex JS** as an interface for SQL queries and PostgreSQL, allowing us to write database queries in JavaScript instead of raw SQL and create our own backend API routes (**prototype/development**)
* **Google cloud** to host the server in the cloud.

Of course, the web pages themselves are composed of HTML with some JS scripts running on the client and server, and we used CSS for styling.

**Changes that took place after submitting the proposal**

Due to time constraints, we had to make some changes to the technology stack:

* **Firebase** as the cloud hosted database (**production**). As a result, we did not have to create custom API routes, but instead just configured the backend to point to the cloud and imported functions that allowed us to make queries to the firebase database
* **Netlify** to host the web application. After compiling the application into a single optimized *build* folder, we simply uploaded the folder to Netlify and customized the website’s URL.

**Group Organization**Scrum Master: Hezron Perez  
Product Owner: Martin Sanabia  
Front-End Developers: Marco Maldonado, Martin Sanabia  
Back-End Developers: Hezron Perez, Derek Gaura

**Software Architecture Design**



(Web-app Server)

(Database Server)

DB connection String

Client

request

response

34.148.19.16

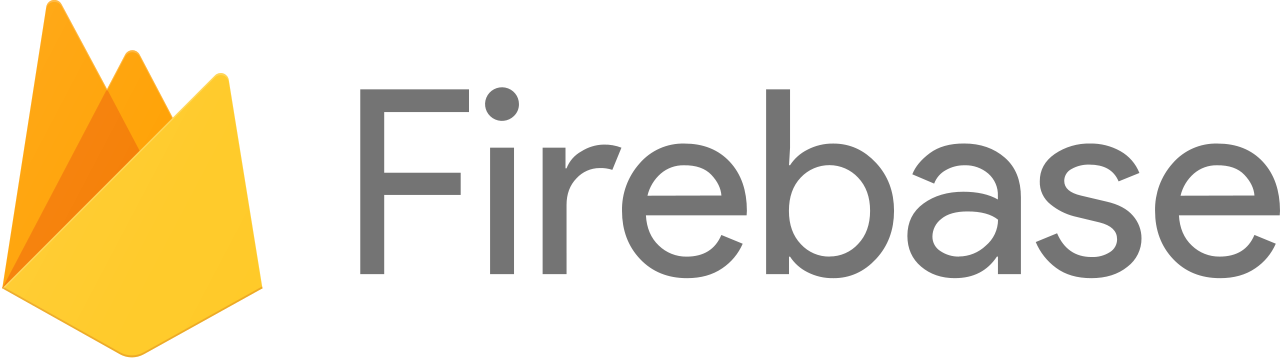
Sees html and css

**JSON**

Serve API!!

Firebase Query

DB info



***Web Application Server***

* From an IP address provided by Netlify, serves a “build” folder, containing the css/html/javascript web components.

***Client***

* Will be given a token after logging in
* Will need this token for the remainder of their session for authentication
* Interacting with certain UI components will send request containing a json object to the web-app server

***Database Server***

* Firebase hosts database documents
* Contain information on the structure of users, carts, items, and orders, as well as specific instances of these data representations

**User Interface Design**

We decided to implement a retro-gaming theme, as our target audience are gamers. The home page has an animated background along with the site’s logo. On that home page is a link that opens the store page.

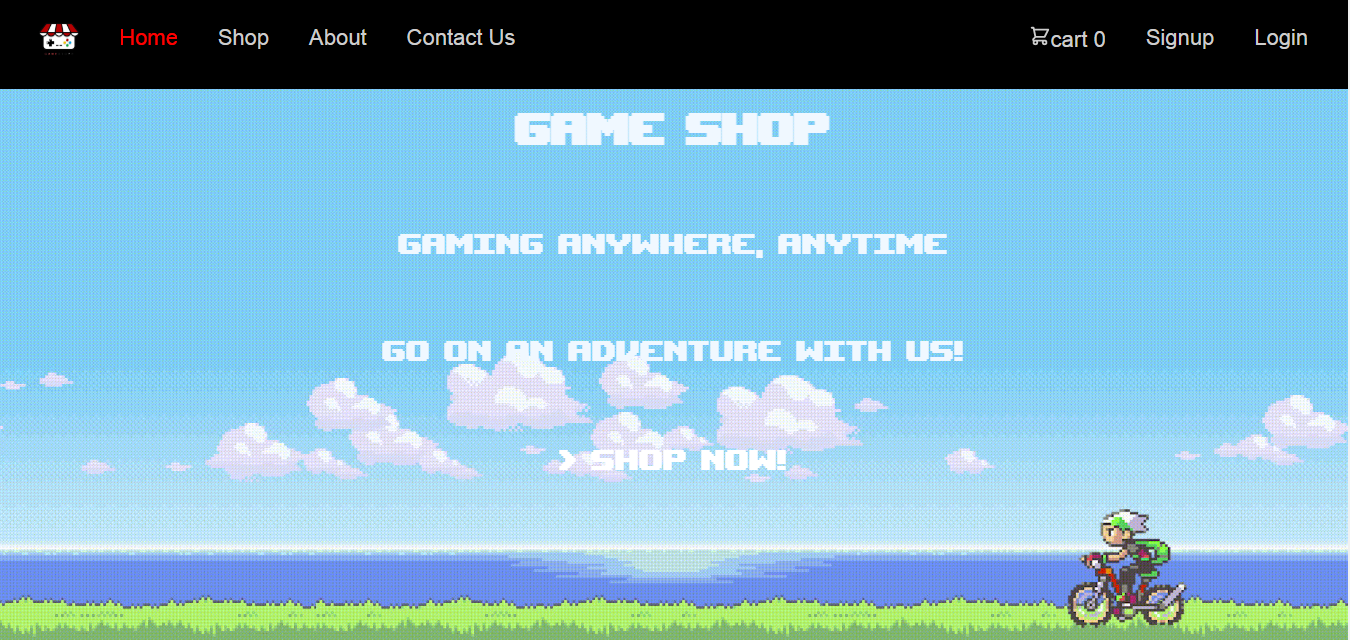
On the store page itself products are displayed in their own box, with each box containing the title of the game, an image, the price, a description and an add to cart button. At the top of the page is a navigation bar with several links to other pages. These pages include an about us page, a link to the store, a contact us page, the user’s cart, and the user account.

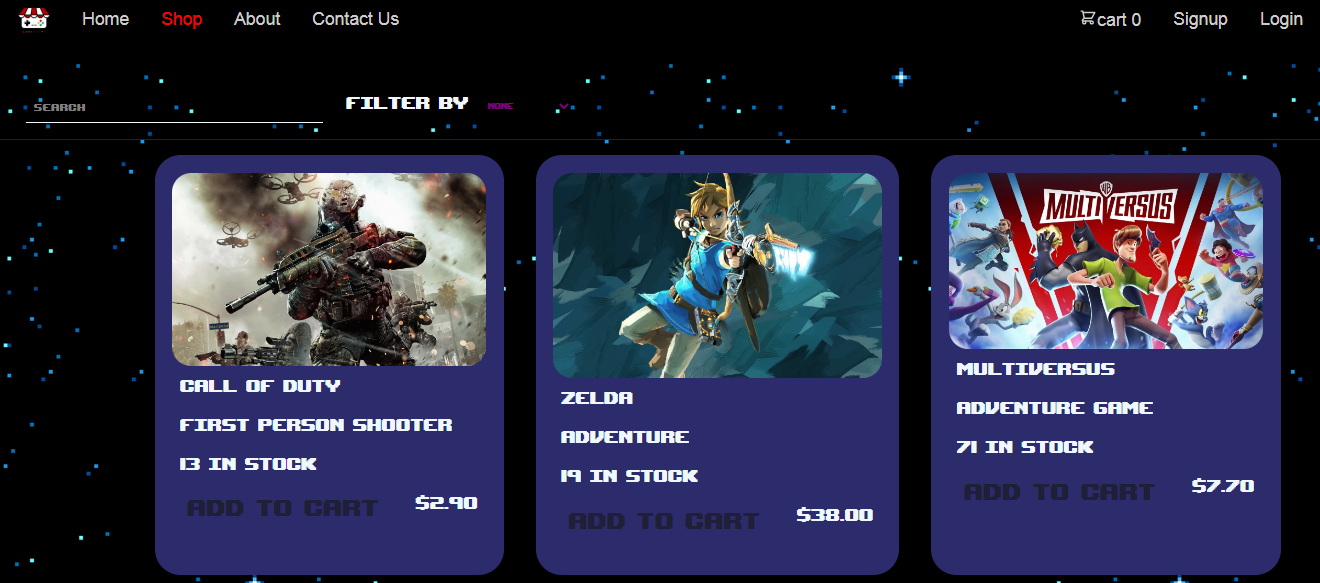
If the user selects the account section they are greeted with a log in or sign-up page. If they log in to a pre-existing account they are returned to the shop and their name is placed on the top right of the screen. If the user creates an account their name is also placed in the top right of the screen.

On the store page clicking the ‘add to cart’ button adds the item to the user’s cart, it also adds a number to the image of the cart next to the user’s account.

In the user’s cart the items in the cart are displayed along with their prices. The user can then increase or decrease the purchase quantity from the cart itself. The total price and the sales tax is also displayed.

Link to database tables: [here](https://github.com/ArcEnig777/TheGameShop/blob/main/backend/migrations/20221021164758_real-db1.js)

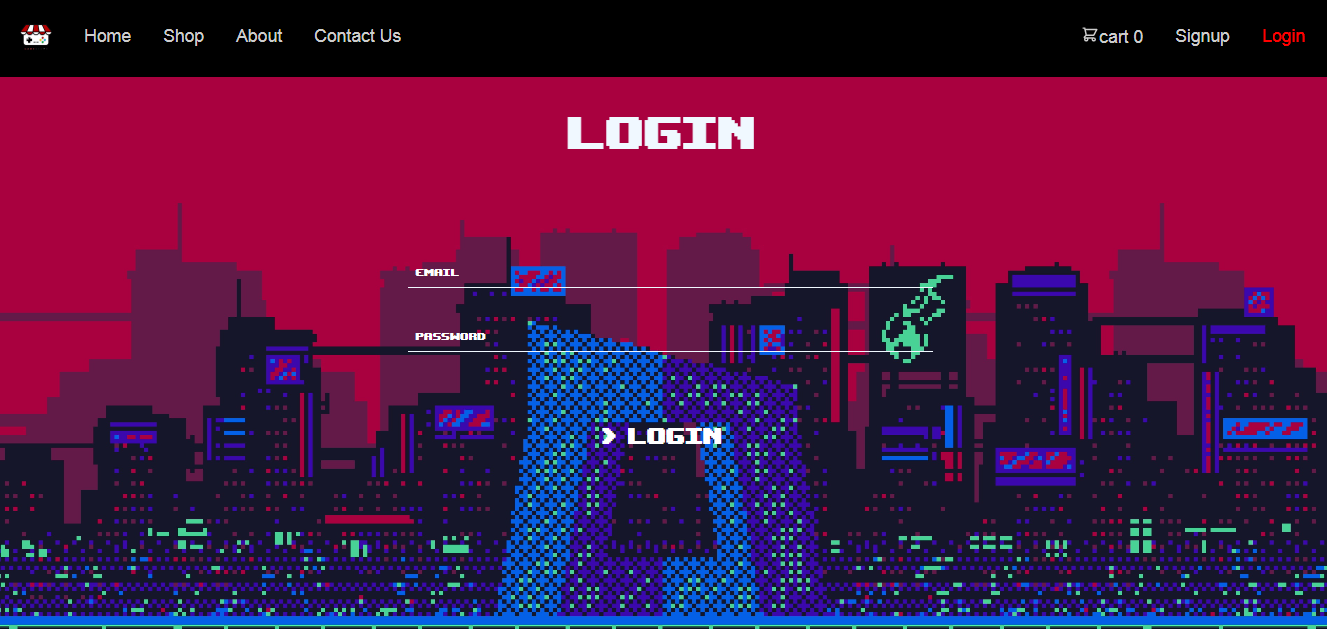
  
*Home page*



*Shop page*



*Contact page*

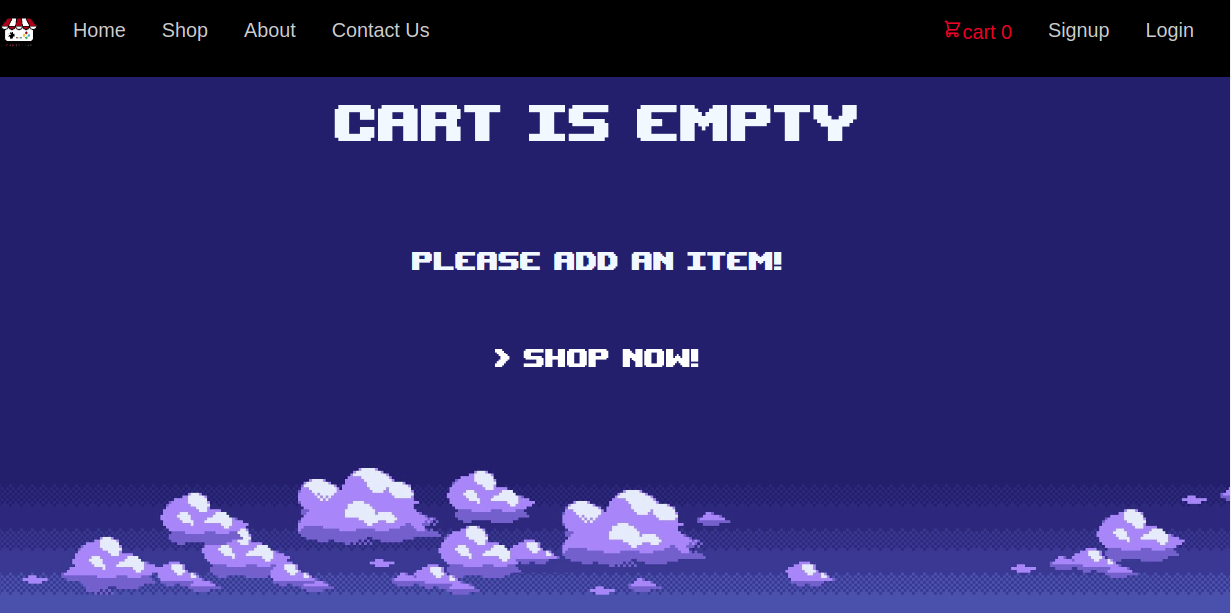


*Login page*

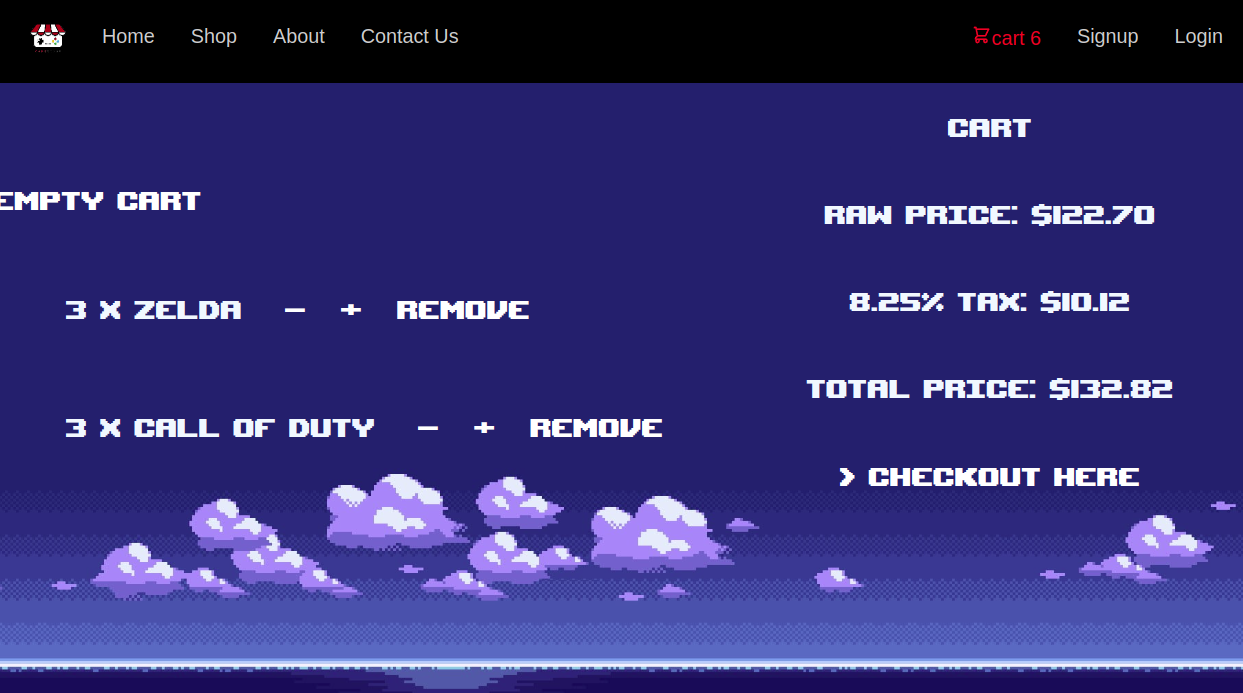
*Admin Page (create a game)*

*Admin Page (edit a game)*

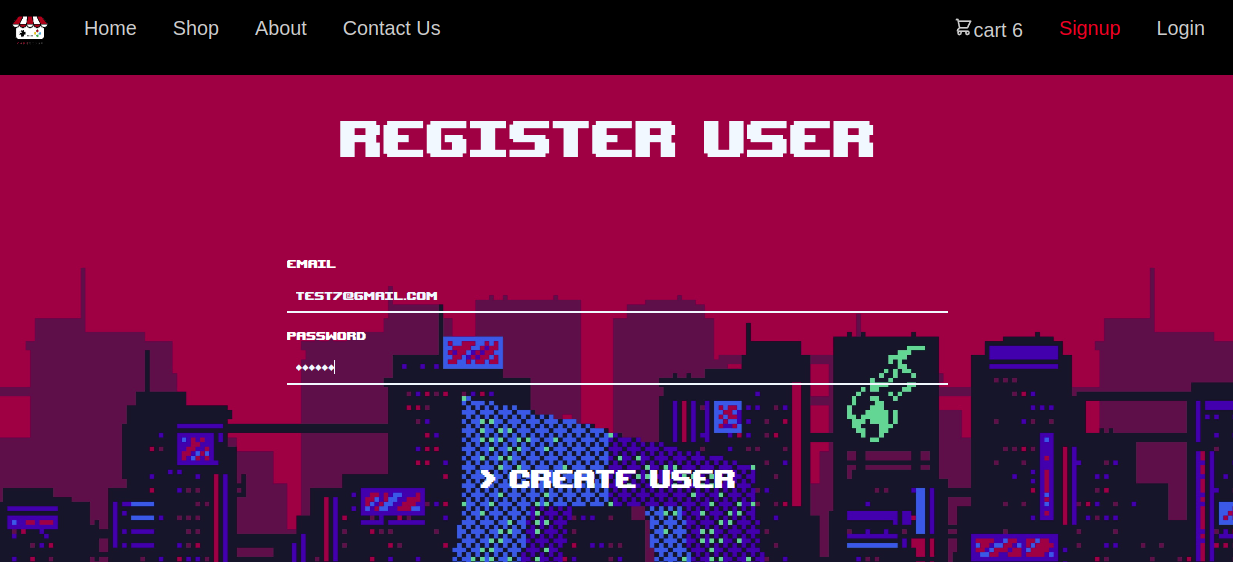
*Admin Page (orders)*

**

*Cart page (with 0 items)*

**

*Cart page (with multiple items in cart)*

**

*Sign up page*

**

*Orders page*

**Product RoadMap: Completed**

Things we did get to on the product road map:

* Hosting local Database (prototype)
* Creating tables for database (prototype)
* Migrating tables from js file to local database (prototype)
* Creating the seeders files and seeding the database (prototype)
* Creating the API routes for CRUD functionality for all database tables (prototype)
* Creating fetch calls to connect front end to backend database (prototype)
* Firebase is backend as a service that we use to host the database in the cloud
* Final product is cloud-hosted by Netlify: <https://thegameshop.netlify.app/>
* Asynchronous JS functions that make calls to the backend
  + Utilize these functions within the html components to update the frontend
* Navbar that returns a different html/js component for each option/tab
* Throughout the website
  + GIFs as backgrounds
* Shop
  + Fetch items from database and display them
  + Sort by price
    - High to low
    - Low to high
  + Sort by quantity
    - High to low
    - Low to hight
  + Different text for first item added to cart
  + Prevents user from buying more items than are available in inventory
  + Makes calls to backend and updates database (user’s cart)
* About
  + Team info
* Contact Page
  + Informative text that separates us from our competitors
* Login
  + Does nothing if login failed
  + Goes to shopping cart on successful login
* Signup
  + Does nothing if signup failed
  + Goes to shopping cart on successful signup
* Signout
  + Goes to homepage upon sign out
* Cart:
  + Ability to increase and decrease items
  + Ability to remove items from cart
  + Displays raw price, taxes, and total price
* Orders
  + Displays list of orders for user currently logged in
* Admin
  + Animated text input area
  + Item creation
  + Item edit
  + Item deletion
  + All orders displayed

**Frontend to Backend Example**

Github repository for locally hosted prototype: [here](https://github.com/ArcEnig777/TheGameShop)

Note: the Github repository for cloud hosted final product is currently private

1. run `yarn start-dev` in the `backend` folder

2.1 run `npx knex migrate:latest` in the `backend` folder

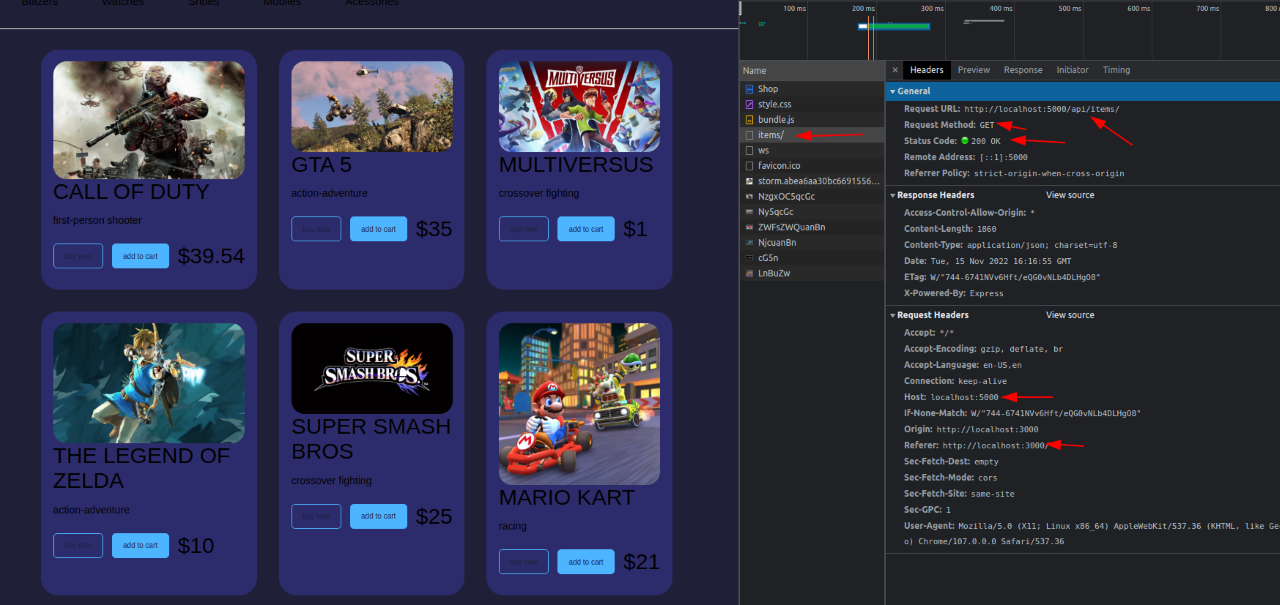
2.2 run `npx knex migrate:down` in the `backend` folder

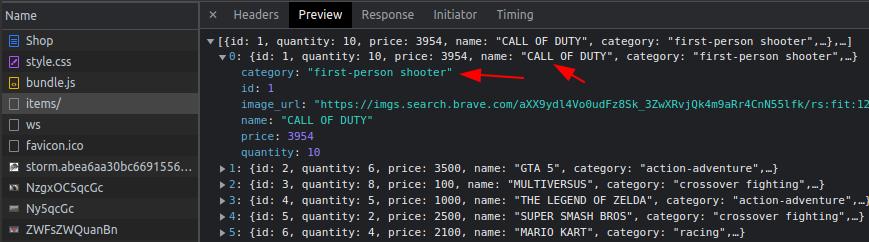
2.3 run `npx knex migrate:up` in the `backend` folder

2.4 run `npx knex seed:run` in the `backend` folder (\*populate Database with values)

3. run `yarn start-dev` in the `frontend` folder to start serving web application

*The web application uses fetch calls to backend altshop.js*





**Appendix**

***Sprint Backlog***  
Features that were on the product road map, but did not escape the sprint backlog:

* Light/dark mode
* Multiple languages
* Captchas
* Rate limiting
* Item limiting
* Bidding on limited items
* Game ratings
* Game age recommendations
* Sorting orders by price
* Sorting orders by date
* Discount Codes
* The ability to place items on hold with a zoom meeting and get ahead of bots:
  + Scalpers use bots to purchase items inhumanly fast before any legitimate uses are able to, so instead we allow users to send a personal email that also verifies their identity - for those who desperately need a game
  + “Do you really want a specific game? Contact us directly and we will personally meet with and try to secure you the game. At the Game Shop, we believe those who really want a game, should be able to get the game. “ A game secured on this way will not go on for bidding.
  + Good service will separate us from our competitors.

***User stories***

* As someone who really wants a game and is tired of it always being out of stock, I want a way to secure a game without pouring in extra money because I am not rich.
  + Feature: Contact page to secure a zoom meeting
* As a parent, I want to be able to select are both appropriate and enjoyable for my kids
  + Feature: Ratings and Genres
* As someone who is new to gaming, I want a variety of games in different genres that are cheap to get
  + Feature: Sort by Price
* As an expert gamer, I want to be able to contribute to the “Game Shop” community
  + Feature: Contact page
* As someone who likes indie games, I want to be able to request that new games be added to the platform.
  + Feature: Contact page