Product Checker Documentation

Andrew Lisi

Design Project I

Contents

Introduction

This system is being designed as part of the CSIS3126 class at Johnson and Wales University to satisfy the requirements for the course. The task is to design, build, test, and implement a software system of our choosing.

The project that has been decided on will be an ecommerce monitoring tool that will allow users to configure alerts on items being sold on a pre-determined set of ecommerce platforms. While the intent is to target products that have been perpetually sold out such as next gen graphics cards and gaming systems, the system will have the ability to monitor any product on these ecommerce platforms and report when it is in stock and available for purchase.

Background

2020 has seen an extreme demand on a variety of electronics which has led to many popular products being sold out or scalped at extremely inflated prices. There have been several major factors that have led to this issue. The first was the Covid-19 global pandemic that saw the world begin to work from home in mass. This put high strain on technology manufactures as many people were not previously equipped to work from home. People also found themselves with much more free time at home so many turned to PC building and video games to pass the time. Additionally, the supply chains themselves have also seen reduced output due to the virus.

Finally, due to the leap this current generation of Nvidia and AMD graphics cards made, they have been in extremely high demand by crypto minors as they are always looking for the most powerful and efficient graphics cards to mine crypto currency.

Purpose

The purpose of this software system will be to help combat scalpers and enable users to buy products at MSRP. To do this the software will monitor products that the end user chooses and let them know when they are available for purchase via an alerting system. The end user will then have a small window where they may be able to buy these products at MSRP before they are inevitably sold out again.

Assumptions

- The chosen programming tools are free and available to download
- This type of monitoring and web scraping is legal
- The use of Firebase will fall within their free tier

Constraints

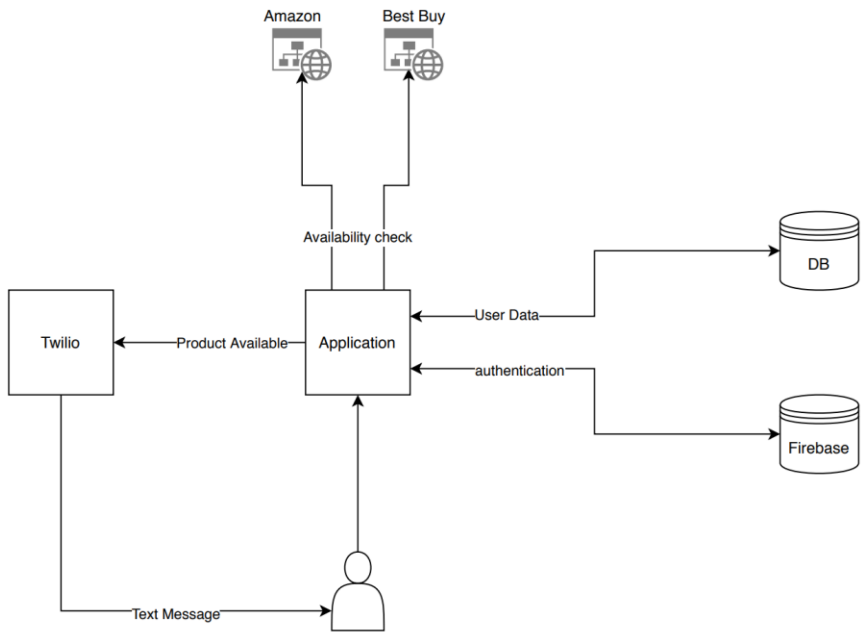
- Project must be completed by end of semester
- Project must be completed at zero cost

Functional Requirements

Users must be able to create an account and sign into their account. Within each users account they will have the ability to configure the system to monitor product availability on Amazon.com and BestBuy.com. The System will save the users products and continuously monitor their availability every 10 seconds until the user removes the product from their list of monitored products.

In addition to the UI showing when a product has become available, the user will have the ability to configure SMS alerting which will send the user a text message if a product has been found in stock.

The system will also keep history of items when found in stock and the price they were available at. If a user chooses a product that the system already has a record of, it will show the user the in-stock dates and related price history.

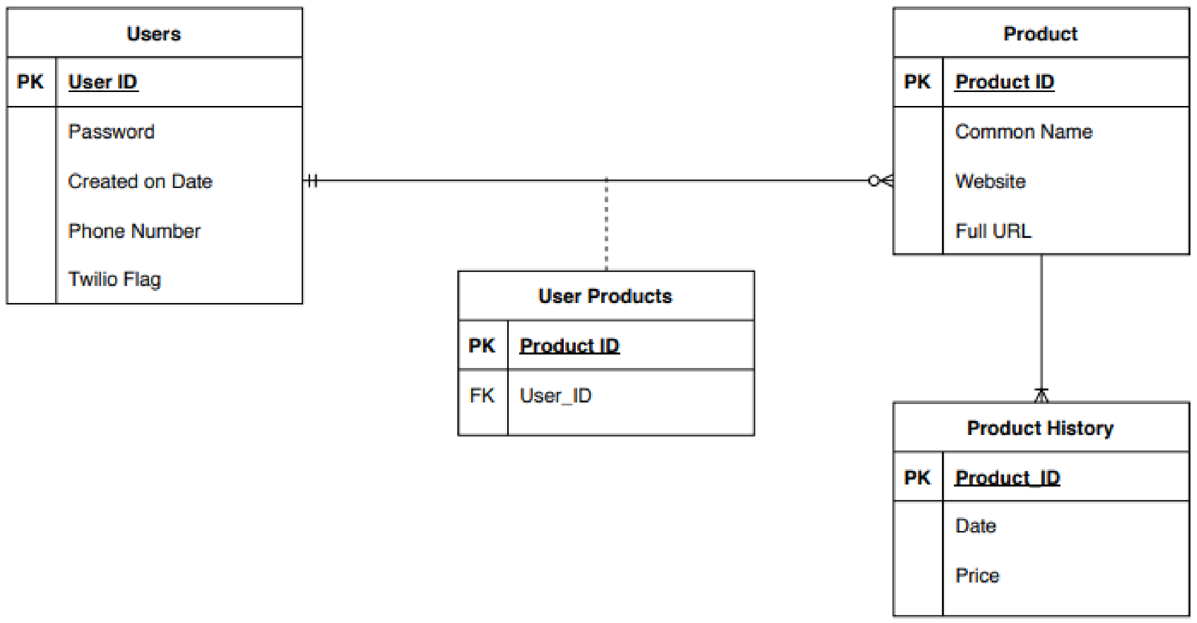


Additionally, the system must include the following requirements:

- At least two platforms – eg. client-server
- A database
- At least one industry standard encryption or hashing algorithm
- At least one object-oriented language component
- Some level of industry standard user authentication.
- In-app context help and documentation. (Exception: real-time system application)
- In the case of a real-time application, a demonstration appliance must be built.

ERD

The Data Model for this system will actually be pretty simple. It will need a Users table to store all the users, their passwords, and some basic preferences. There will also be a product table which stores any unique product along with it's common name (For Display purposes), it's website (Amazon.com or BestBuy.com) and then the full product URL. There will be a Product History table for historical views of when a product was last available and the price. Finally, there will be the User Products table which will store all Products that a user wishes to monitor.



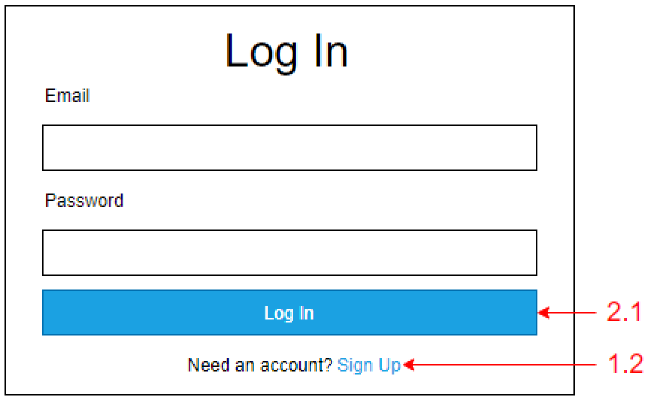
UI

Red numbers indicate the corresponding page it will bring you to

1.0 Authentication

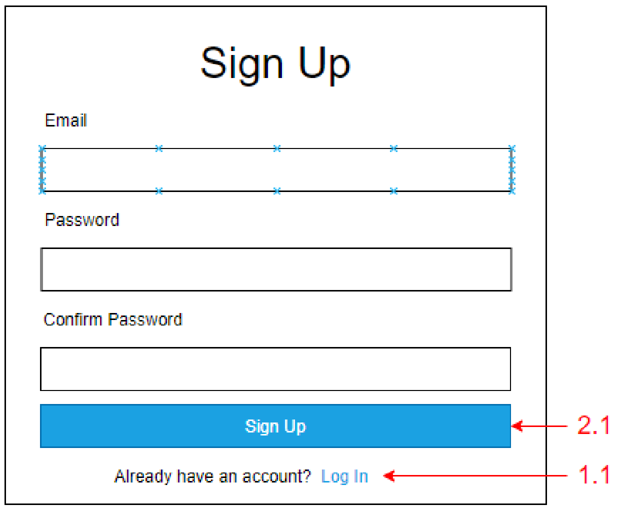
1.1 Log In page

Initial Page will show Log In screen with fields for email and password. The ability to Sign Up for a new account is also included here.



1.2 Sign Up page

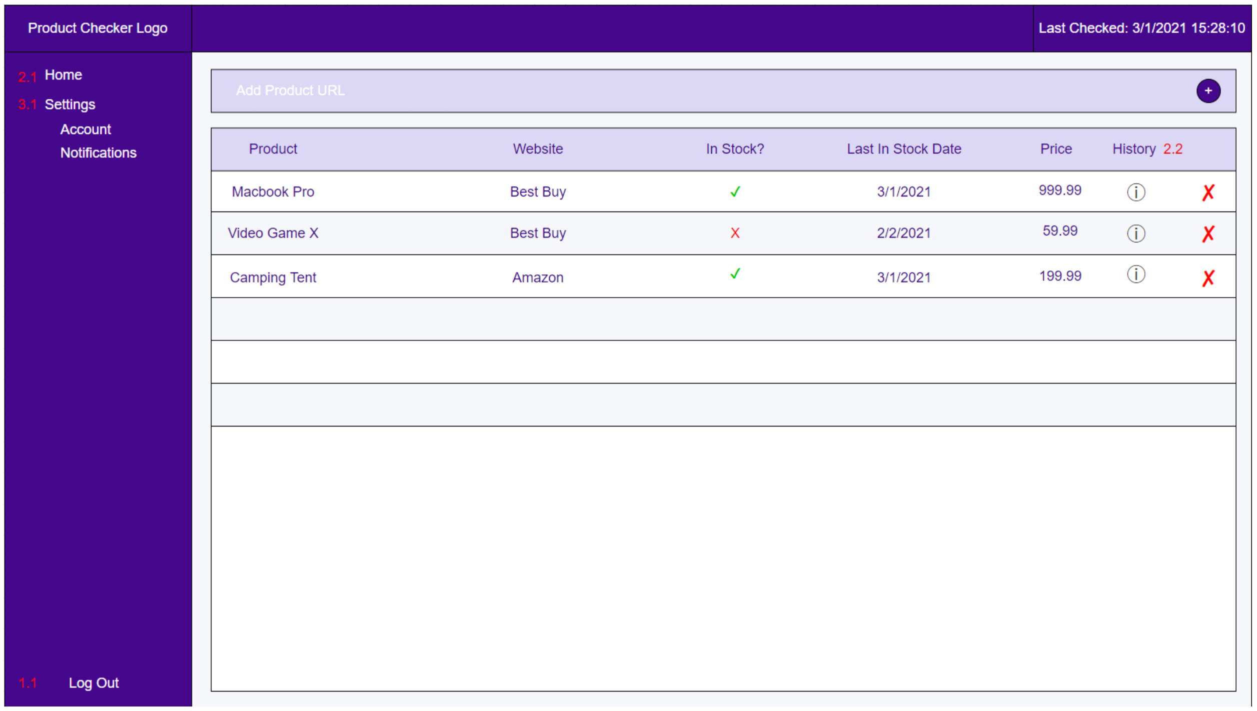
Here users can sign up for a new account which will require email, password, and password confirmation. An option to go back to “Log In” has been included in the case a user has selected this on accident.



2.0 Main Application

2.1 Home Page

A fairly simple and hopefully intuitive Home Page will also act as the main page of interaction for users. They will have a simple ADD URL bar at the top which will add Products to their User Product table. It will then parse the URL checking for validation and supported websites(BB and Amazon). If it passes validation it will be added to the User Product Table. If it has not been included in the Products table yet, it will also be added there. Users will have an easy way of removing a product from their list as well as checking the history with the "i" icon.



Product	Website	In Stock?	Last In Stock Date	Price	History 2.2	
Macbook Pro	Best Buy	✓	3/1/2021	999.99	ⓘ	✗
Video Game X	Best Buy	✗	2/2/2021	59.99	ⓘ	✗
Camping Tent	Amazon	✓	3/1/2021	199.99	ⓘ	✗

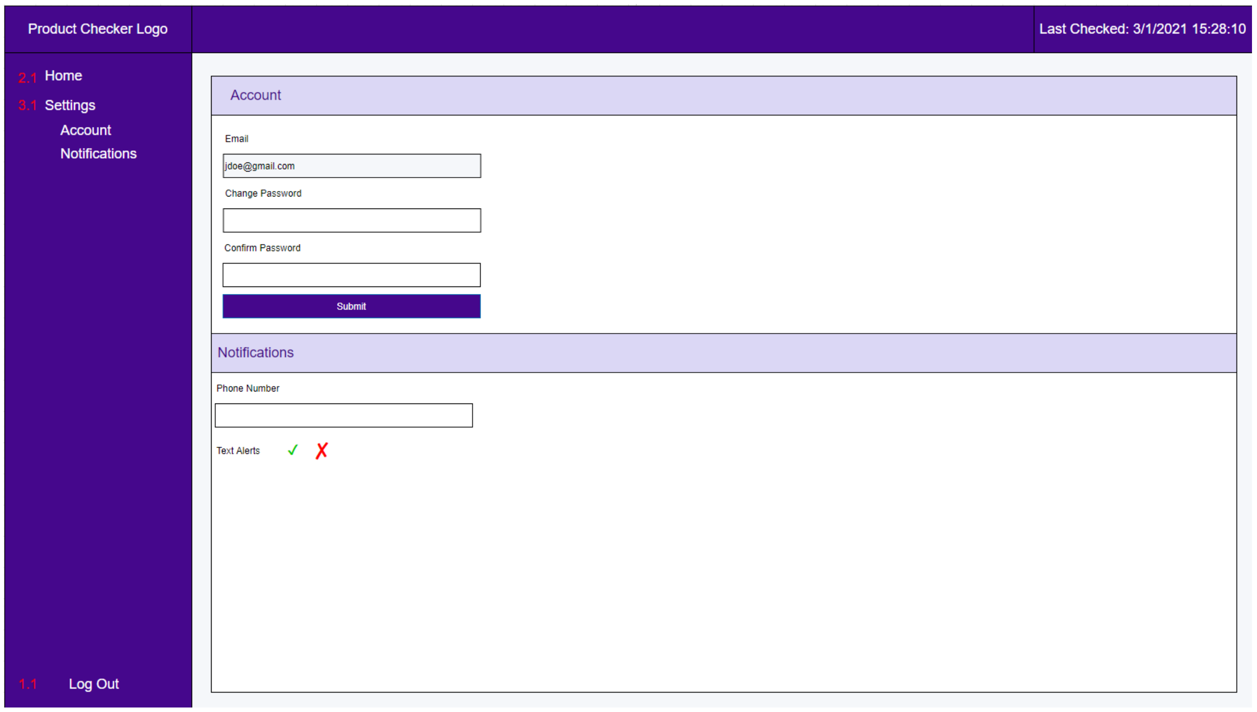
2.2 Product History

*Not shown, but it will be a pop-up style Historical graph with dates on the x axis and price on the y axis. Will show last 1 year of history.

3.0 Account

3.1 Settings

Settings will be kept relatively simple with the ability to Change Passwords, and configure text notifications. Later versions of the product checker will also include email notifications, discord notifications, and the ability to integrate with SSO like Google, Facebook, or Twitter for authentication.



Product Checking Engine

When a user adds a product to their collection of products, it will add it first to the User Product table with the user's ID and the Product ID, then it will see if the product exists in the Product table. If the product is not already in the product table, it will add it to the table. If it already exists the in the product table, it will not perform any further actions.

The main Product Checking engine will cycle through every product in the product table and check the website to see if it is available. It will output the results of this check to the Product History Table.

When a user is looking at their main page/ home page the UI will automatically update the In Stock?, Last in Stock Date, and price attributes for any product shown. Clicking on the "i" icon for any given product will show you a historical chart for the product with price and availability.

Because this product is targeted towards users who are searching for items that are mostly out of stock, a decision has been made to hardcode a pretty frequent 60 seconds check on each product to update as close to real time without putting extreme load on the system. More frequent checks than that will get very resource intensive when the product table starts to grow, and less frequently than that will start to defeat the purpose of getting alerts when hot products become available.

Version 0.2

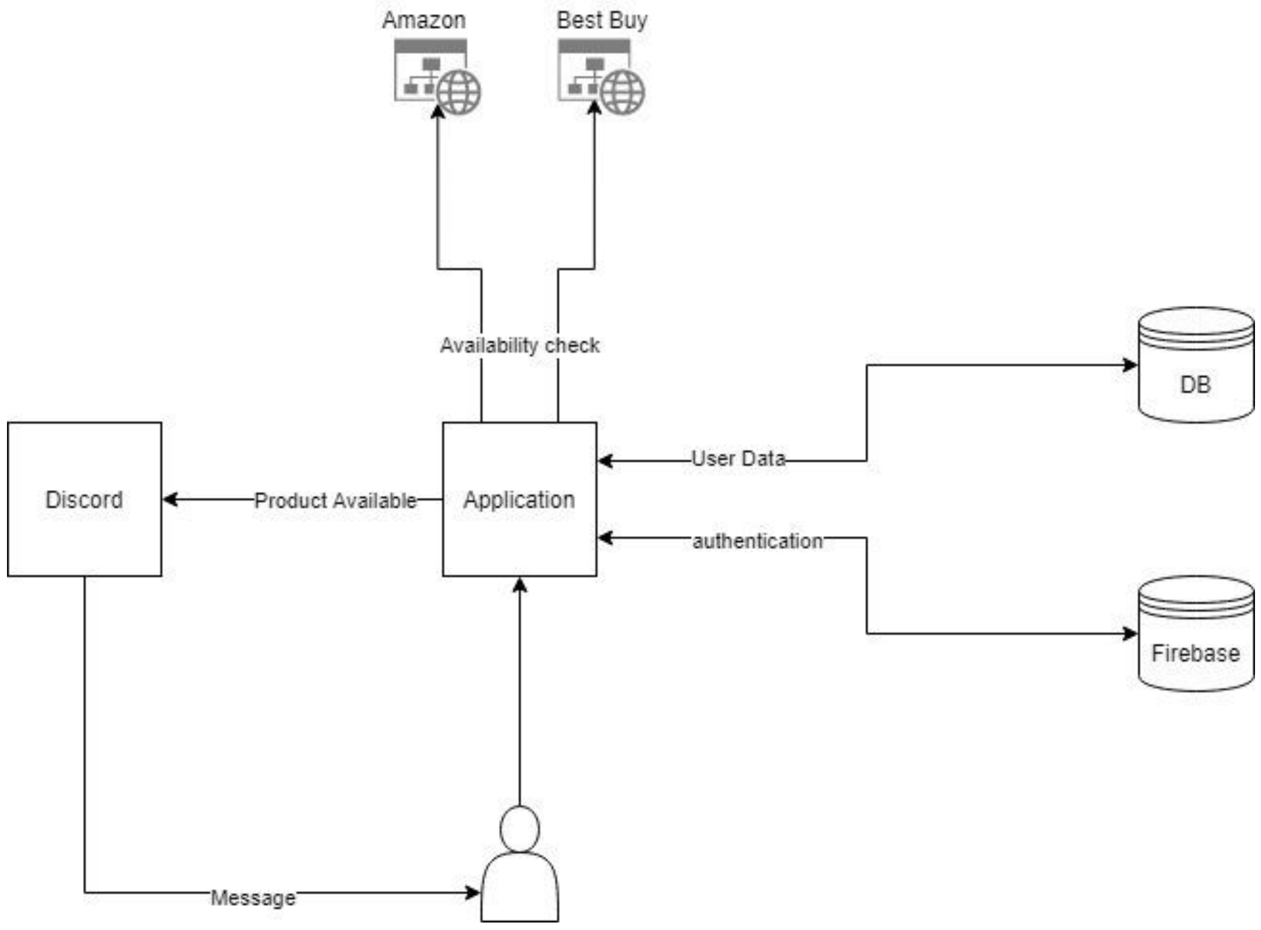
Throughout the design and implementation phase several stack and architectural changes were made after careful thought and consideration of the use case of this project. Some of the first challenges I came across were my lack of front-end experience. Trying to learn React felt like a stretch when I had never used HTML, CSS, or JS. Instead I decided to move to a Flask application and use Bootstrap to help me created my elements. The Flask decision came because I am looking to get into more backend-oriented programming and wanted to expand on my Python and object-oriented skills. This ultimately proved the right decision for me as I learned so much more about Python and object-oriented programming. Bootstrap also made it much easier after a quick crash course in HTML and CSS I was able to pick up Bootstrap easily and make what I think is a pretty decent looking front-end.

The second major change was more architectural as I thought about how I wanted this project to live on after this course, and ultimately realized I would like to distribute it for free knowing there are several communities who would appreciate it's use. Because I was not willing to pay for hosting and Twilio costs for other users at this point, I changed the architecture to be more efficient for users to run it on their own machines. This led to some changes in the ERD as well as swapping out Twilio for Discord notifications. I felt like asking users to create their own Twilio account and enter their API key would be asking too much and Discord is an extremely popular messaging application that many users in my targeted demographic already have. It also has a mobile and desktop application giving people further flexibility on how they will receive notifications.

The final major change due to this rethought architecture is that I incorporated the Product Checking Daemon directly into the package instead of its own completely separate service. Upon application initialization, a separate thread is called to start the product checking that runs in a loop. This loop time can be defined by the end user in their Account page.

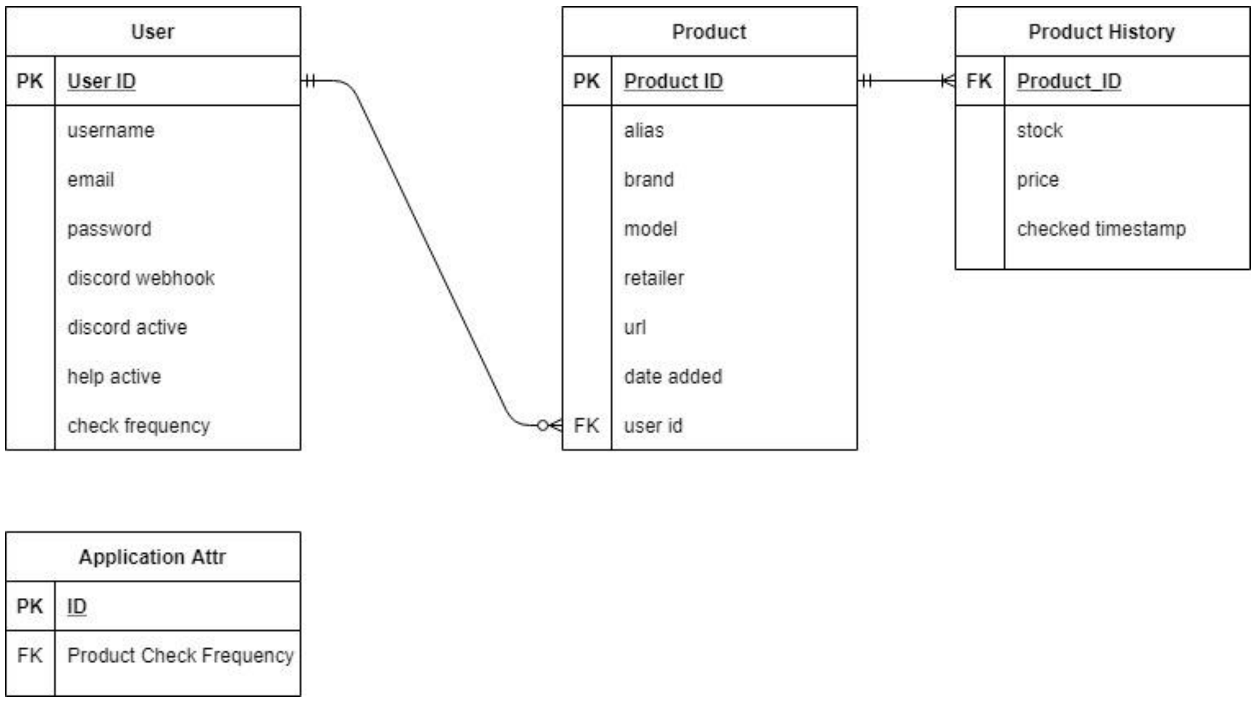
Architecture update

This shows the updated architecture with Discord acting as the notification service. Upon a Product which was previously out of stock, becoming in stock, a formatted message containing the Alias, URL, stock status, and price is sent to the user via Discord Webhook.



ERD Update

After the decision to design this application on a users personal computer as opposed to a central server, I decided that the User-Product many-to-many table was no longer necessary. I also added the Application Attribute table to hold application wide settings. This was necessary due to the multi-threaded design and the inability in Flask for threads to share data. This was they can read and write to the database any information they do need to access.



Test Plan

I created a comprehensive test plan that Is designed to test all of the possible user inputs and user actions. Ideally If we had more time or I had a bigger team, I would have loved to implement Selenium to automate testing or Unit testing, but for now a manual process was developed. All user form fields have validation and expected lengths and most have allowable characters or checks for correct parameters (ie amazon or bestbuy in url for "add product")

Login page

Username Field

- String Testing
- Length Testing
- Known usernames
- Unknown usernames

Password Field

- String Testing
- Length Testing
- Correct Password for account
- Incorrect password for account

Login Button

- Upon correct credentials it should authenticate user and direct to Dashboard page.
- Upon incorrect validation, it should inform user and make them try again.

Forgot Password? Link

- Make sure it successfully sends email upon found accounts
- Make sure it does nothing if account is not found.

Sign Up Link

- Should redirect to Registration page

Registration Page

Username Field

- String Testing
- Length Testing
- Available usernames
- Already taken username (must be unique)

Email Field

- String Testing
- Length Testing
- Available email
- Already taken email (must be unique)

Password Field

- String Testing
- Length Testing

Confirm Password Field

- String Testing
- Length Testing
- Matches Password field.

Sign Up button

- If validations fail, it should stop and explain which fields failed.
- If validations pass, it should bring user to “Add product” page to get started and display welcome messages.

Sign In Link

- Should redirect to Sign In page.

Dashboard Page

Dashboard should correctly display all Table Data including Alias, Brand, Model, Retailer, Stock, Price, Checked, History button, and delete button.

Dashboard Help Button

- Click the help button launches the Help Modal for the Dashboard Page.
- Modal closes properly.

Alias field

- Alias is a hyperlink that brings to the product URL.

History button

- Redirects to Graph Page.

Delete button

- Pops up a modal and asks user to confirm deletion of product.
- Upon deletion, page refreshes and table row is removed from Dashboard.

Add Product Page

Add Product Help Button

- Click the help button launches the Help Modal for the Add Product Page.
- Modal closes properly.

Product Alias Field

- String testing
- Length testing
- Validations working properly and showing messages to users

Product URL

- String testing
- Length testing
- Supports only Amazon or Bestbuy
- Upon link not matching *amazon.com* | *Bestbuy.com* fail validation

Submit Button

- Upon failure of validations informs user and allows reattempt.
- Upon Success, grabs additional product attribute from retailer website, appends to the product record and inserts product to Database and redirects to Dashboard.

My Account Page

Account Help Button

- Click the help button launches the Help Modal for the Account Section.
- Modal closes properly.

Username Field

- String Testing
- Length Testing
- Available usernames
- Already taken username (must be unique)

Email Field

- String Testing
- Length Testing
- Available email
- Already taken email (must be unique)

Password Field

- String Testing
- Length Testing

Confirm Password Field

- String Testing
- Length Testing
- Matches Password field.

Preferences Help Button

- Click the help button launches the Help Modal for the Preferences Section.
- Modal closes properly.

Product Check Frequency

- Field value in in specified range.
- Value actually updates the checking frequency.

Notifications Help Button

- Click the help button launches the Help Modal for the Preferences Section.
- Modal closes properly.

Discord Field

- Accepts correct URL’s
- Fails validation upon URL’s that don’t match *discord.com*
- Length Testing
- Removing URL where previously there was one, automatically toggles alerts to off.

Discord Alerts

- Toggling On/Off actually turns notifications on or off.

Submit Changes Button

- Any failed validations show on respective fields
- Upon success, actually writes values to database.

Various additional functionality

Site Navbar

- Dashboard link directs to Dashboard Page.
- Add Product link directs to Add Product Page.
- My Account link directs to My Account page.
- Logout Button logs user out and de-authenticates.

Product Checking Daemon

- Successfully launches at application start
- Checks all available products, updates, and writes to database
- Sleeps for the amount of time defined by User Product Check Frequency which is written to appattr table.
- Writes output to terminal when starts check, completes checks, and for how long it will sleep.

Additional Documentation

All of the technical documentation including Module, Class, method, and function docs can be found hosted on the Github Repo at https://ixisunnyixi.github.io/product-checker/

Windows Installation Instructions

*The following assumes Python 3.9.4 is installed. If you do not have it, it can be found at https://www.python.org/downloads/

- Download the code at the repo either by closing the repo or downloading the zip file. https://github.com/IXISunnyIXI/product-checker

- Extract the contents to a directory of your choosing.

- Using command prompt change to specified directory: cd <enter/full/filepath/of/extracted/contents>

- Create a virtual environment: py -m venv <venv or name of your choice>

- Activate virtual Environment: <venv name>\Scripts\activate.bat

- Install requirements: python -m pip install -r requirements.txt

- Start application: run.py

- Using a web browser, type in the URL bar: http://localhost:5000/register

Enjoy!