Project: E-Commerce Web Scraper

COMP 3100: Web Programming

Iteration 2

Team #12
Imtiaz Anik
Joy Kumar Roy
Shawon Ibn Kamal

Introduction

Functionalities

The main functionality of the application is that it scrapes product information from e-commerce websites particularly from Amazon and Best Buy. In our first iteration we had mentioned that our web scraper would be based on scraping Walmart products; however, as we were trying to work with Walmart data, we figured that the content is loaded dynamically and the data is scraped slowly. That is why we have tried to work with Amazon and Best Buy data and we were able to successfully do it.

It is used to scrape data from either of the websites using product url and store them in a database. The user can search and display the products stored in the database. The user will have a chance to delete and update any product as well.

Some of the technologies used in implementing this project is ExpressJS as the backend framework, MongoDB as the database, Mocha as the unit testing library, Mongoose to implement object modeling, Passport for authentication, puppeteer and cheerio for web scraping, request and axios.

The functionalities implemented for this project includes user signup and login, search products that are in stock and out of stocks using product name. We can also use sku to search for a product. We can directly update any product form the website if the price or any other info is changed, search products from the ones stored in the database. We can delete a product and can add a product manually from the application. All the functionalities can be accessed from the dashboard which will be built in the front-end of Iteration 3.

Users can log in to their account by giving their information like username and password. A user can also edit their account or delete it, in case they want to. If the user is using the Web Scraper application for the first time, they need to sign up by providing their information like password and email.

Models

In our project we used a npm-package called ‘mongoose’ to implement the models. The models provide an abstraction from pure mongoDB and allow CRUD operations on databases easier. It also provides validation for the field types to make sure users do not miss out on required fields or insert values of wrong types.

1. User

The User model has three fields:

- name:string
- email:string
- password:string

The functions we use in this models are

A. User.findOne()

   This function is used to retrieve a user from the database given the unique id or email.

B. User.findOneAndUpdate()

   This function is used to update an existing user.

C. User.deleteOne()

   This function is used to delete an existing user.

D. User.find({})

   This function can be used to find all users or multiple users with the condition passed in.

We will not use the model to create any users as we are signing up using the passport npm-package.

2. Product

The product model has these fields:

- title:string
- newprice:string
- oldprice:string
- newstock:string
- oldstock:string
- sku:string
- company:string
- url:string
- updatestatus:string

The functions used in the model are:

A. Product.create()

   This function is used to submit a new product in the database.

B. Product.findOne()

   This function is used to retrieve a product from the database given the unique id or email.

C. Product.findOneAndUpdate()

   This function is used to update an existing product.

D. Product.deleteOne()

   This function is used to delete an existing product.

E. Product.find({})

   This function can be used to find all users or multiple products with the condition passed in.

Routes and Controllers

1. User routes

A. GET: http://localhost:3000/

   This route is used to retrieve a list of all signed up users.

B. GET: http://localhost:3000/:email

   This route returns a user object given the email.

C. PUT: http://localhost:3000/email

   Body: name:string, email:string, password:string

   This route is used to update a user in the database.

D. DELETE: http://localhost:3000/:email

   This route is used to delete an existing user in the database.

E. POST: http://localhost:3000/signup

   Body: name:string, email:string, password:string

   This route is used to register a new user in the database.

F. POST: http://localhost:3000/login

   Body: email:string, password:string

   This route is used to login with valid email and password, which will return a JWT token which can be used to visit secured routes.

2. Product routes

A. GET: http://localhost:3000/search

   Params: sku:string

   This route is used to search a product with the given sku

B. GET: http://localhost:3000/instock

   This route is used to get products that are in stock.

C. GET: http://localhost:3000/outofstock

   This route is used to get products that are out of stock.

D. GET: http://localhost:3000/pricechanged

   This route is used to get products that went through price change.

E. GET: http://localhost:3000/backinstock

   This route is used to get products that are back in stock.

F. GET: http://localhost:3000/updated

   This route is used to get products that are newly updated.

G. GET: http://localhost:3000/notupdated

   This route is used to get products that are not updated.

H. GET: http://localhost:3000/fetch

   Params: search:string

   This route takes in the url of Best buy or Amazon and scrapes the product information from the website.

I. POST: http://localhost:3000/new

   Body: title:string, price:string, stock:string, url:string: sku:string

   This route pushes a new product to the database with the information given from the body of the form.

J. POST: http://localhost:3000/update

   This route automatically updates all the products that are added in the database.

Tests

We have used MochaJS to test the methods. 2 separated files named tests-users.js and tests-products.js have been created in the tests directory. The tests in the tests-users.js file test the methods in the users.js and the tests in the tests-products.js test the methods in the products.js. There are a total of 13 Success and Fail tests in the tests-users.js file and 7 Success and Fail tests in the tests-prorducts.js file.

1. User tests

1. Fail 1: POST - Invalid Email - Signup
   The test is performed to check if the given email is rejected when it is invalid during Signup for a new user.

2. Fail 2: POST - Invalid Password - Signup
   The test is performed if the provided password is rejected when it is invalid during Signup for a new user

3. Fail 3 - POST - Invalid Email - Login
   The test is done to check if the provided email is rejected during Login when the email is invalid.

4. Fail 4 - POST - Invalid Password - Login
   The test is performed to check if the password is rejected during Login when it is invalid.

5. Fail 5 - GET - Invalid Email - getOne
   The test is performed to check if an invalid email is rejected when we search a particular user.

6. Fail 6 - PUT - Invalid Email -updateOne
   The test is done to check if an invalid email is rejected when we try to update an user with a specific email.

7. Fail 7 - DELETE - Invalid Email - deleteOne
   The test is performed to check if an invalid email is rejected when we try to delete an user with a specific email.

8. Success 1 - GET - all
   The test is done to check if all the users are listed using a specific URL.

9. Success 2 - GET - login
   The test is done to check if an user can login using valid user credentials (email and password).

10. Success 3 - POST - signup
    The test is performed to check if an user can sign up to the application using valid email and password.

11. Success 4 - GET - getOne
    The test is done to check if an user is found when searched using a valid email.

12. Success 5 - DELETE - deleteOne
    The test is performed to check if an user of a specific email can be successfully deleted.

13. Success 6 - PUT - updateOne
    This test is done to check if the user details of an user can be updated using a different email and password.

2. Product tests

A. Success 1. POST - Valid product

   This tests if the application can add a product by doing a post request to /product/new

with valid data.

B. Failure 1. POST - Duplicate product

   This tests if the application successfully rejects a product submission if the product is already been added to the database.

C. Success 2. GET - Valid product

   This tests if an inserted product can be retrieved using a GET request to /products/search using sku as the parameter.

D. Failure 2. GET - Invalid product

   This tests if the application gives a proper rejection message if the product sku does not exist in the database.

E. Success 3. POST - Update all product

   This tests if the application successfully updates all the product automatically when a post request is made to /products/update

F. Success 4. Delete - Valid product

   This tests if the application successfully deletes a product when a delete request has been done to /products/delete.

G. Failure 4. Delete - Invalid product

   This tests if the application successfully handles the delete request if invalid sku has been provided.