Assignment #08: Connoisseur

Background:

Staying on top of the products at the Liquor Control Board of Ontario is a high priority for any serious Canadian. We're going to make it easy to get information regarding LCBO products, stores and inventories using the LCBO API (application programming interface).

An API is an interface implemented by an application which allows other applications to communicate with it. When you combine data or functionality from more than one application to create another application, you have essentially created a mashup.

Hypertext Transfer Protocol (HTTP) is the life of the web. It's used every time you transfer a document or make a web request, but HTTP is rarely fully understood among web developers. There are a set of web architecture principles, known as REST, which are closely related to HTTP, that establish a common set of actions to interact with any web device or operating system.

Learning Goals:

We're going to build a simple application that grabs products from the LCBO API. We will have two files: router.rb and html_generator.rb. The goal of this exercise is to become familiar working with third-party APIs. We'll use **ARGV** to accept command line arguments when executing a Ruby script. We will use our script in combination with the command line arguments to retrieve specific data from an external API and use that data to generate HTML pages.

This will all be processed by a router so you will need to understand what a **router's purpose** is and what **JSON** is and how to parse it. You'll need to understand what the gem **"open-uri"** does and use object oriented principles (OOP) to DRY up your code and establish singular responsibilities for related sections of your program.

External Resources:

- Rails Guides - Routing
- Stack Overflow - What's the point of ARGV in Ruby?
- Stack Overflow - Parsing a JSON string in Ruby
- A Beginner's Guide to HTTP and REST
- CRUD / REST Infographic

Suggested Process:

Implement a Router

1. As a user, I have a router.rb file that is run from the command line to output HTML to the console.
2. As a user, I can specify two actions, "index" or "show" + [id], as my command-line arguments.
   a. If the "index" action is specified, the command line should output the HTML for a list of products.
   b. If the "show" + [id] action is specified, the command line should output a detailed product page for a product with the given id. Note that [id] should be replaced with an actual product id (without square brackets). The product id's can be found in the output of the index action.
3. As a user, if I specify "index" + an optional second parameter, which can be treated like a search term, to only show results that match the given word (e.g. "ruby router.rb index bourbon" would list only products that contain the word "bourbon" in one of the fields).

Implement an HTML Generator

1. As a developer, create an HTMLGenerator class to print the structure and content of an HTML page dynamically. The HTMLGenerator class should have the following methods: "index", "show", "print_header", "print_footer", "retrieve_data(url)", "format_price(cents_string)".

Implement CSS

1. As a designer, create a visually appealing design so that the use of the site is more enjoyable.