Find API Design

User Stories

- As a user, I want to be able to search an Alcoholic Beverage, so that I can see which stores carry it.
- As a user, I want to be able to look up LCBO stores, so that I can get location and contact data about them.
- As a user, I want to be able to search an Alcoholic Beverage, so that if I want to go and buy it, I can easily locate the stores that carry it in stock.
- As an admin, I want to be able to update, delete, or create Stores along with location data, so that users get up to date information about existing LCBO stores.

Justification

This API includes 2 endpoints called PyDrink.com/find/stores which returns store ids that have a user provided drink name, and PyDrink.com/find/location which given a store id provide location and contact information about the given store. The two endpoints together create the composite Endpoint PyDrink.com/find which given a drink name will return location data about all stores.

As seen in the table below, PyDrink.com/find/stores and PyDrink.com/find/location both do not require parameters in order to work (will be explained further in the document when looking at how they work) and as such can be used as a regular GET request with parameters added to the URI, however the PyDrink.com/find/location endpoint requires user input and cannot function without it. URI parameters are inheritably optional, and as such a GET request cannot be used as a body is required. The POST operation was used instead since a body can be provided, and since the request performs a "search", I can use the POST verb because I am "creating" a search from the user perspective. I don't need to literally create/write something in a database in order to use a POST.

The rest of the end points are pretty self-explanatory, DELETE is used to Delete, POST is used to Create, and PUT is used to update.

Endpoints

HTTP Header	CRUD Representation	Endpoint	Action	Status Codes
GET	Retrieve	PyDrink.com/find/stores	Retrieve stores that carry a given drink	200, 400, 404
GET	Retrieve	PyDrink.com/find/location	Retrieve location information for a given store	200, 400, 404
POST	Retrieve (Create Search request)	PyDrink.com/find	Retrieve location information for a given drink	200, 400, 404

PUT	Update	PyDrink.com/stores	Update stores catalog given a drink name, and store ids	200, 400, 401, 403, 404
PUT	Update	PyDrink.com/location	Update a stores location information given a store id	200, 400, 401, 403, 404
POST	Create	PyDrink.com/stores	Create a store with a given catalog	201, 400, 401, 403, 404
POST	Create	PyDrink.com/location	Create location information to link to a store given a store id	204, 400, 401, 403, 404, 409
DELETE	Delete	PyDrink.com/stores	Delete stores given their ids	204, 400, 401, 403, 404
DELETE	Delete	PyDrink.com/location	Delete location information for a given store given its id	204, 400, 401, 403, 404

Status Codes Explanation

200 (OK) – This status code represents a successful request. Used for all endpoints that don't create or delete.

201 (Created) – This status code is also representing a successful request with the added context of a created object. In this case, 201 is used when creating stores. A 201 was used since this specific request will return the store id that represents the unique identifier of the store.

204 (No Content) – This status code is very similar to a 201, except it doesn't return a body. This status code represents a successful request and is used specifically when creating location information for a given store. No data is sent back to the caller on success since nothing other than a link between the data provided and a given store is created.

400 (Bad Request) – This status code is returned if the user doesn't provide the required information to the specified endpoint.

401 (Unauthorized) – This status code is returned if an admin only endpoint is requested but no Authorization Bearer Token is provided. This is used for Updating, Creating, and Deleting stores and location information.

403 (Forbidden) – This status code is returned if an admin only endpoint is requested but the provided Authorization Bearer Token doesn't have the privilege of accessing the requested URI resource.

404 (Not Found) – This status code is returned if the requested information to be retrieved or updated is not found. This status code is also returned for creating stores if the provided catalog (drink names) are not found at LCBO (so no information is available about them), for creating location information if the store ids provided cannot be found, or for deleting stores or location information if the provided store id cannot be found.

409 (Conflict) – This status code is specifically used for creating location information given a store id. It is returned if a store id already has associated location information. In this case an Update (PUT) request

should be used instead, or a Delete request to delete location information from a specific store followed by the Create request again for creating location information given the same store id.

Further Explanation of 3 Specific Endpoints

- Most of the endpoints are straight forward and aren't difficult to understand once the three following endpoints are explained (explaining all of them would be too big of a design document and could be an assignment all on its own).

GET PyDrink.com/find/stores

This request takes the optional parameter drink_name, if it is provided, the URI would like like this:

GET PyDrink.com/find/stores?drink_name="Smirnoff Vodka"

If not provided, all stores will be returned otherwise, the following is the response for the above request (Status Code 200):

{

"message": Found 637 LCBO stores that carry Smirnoff Vodka.

"store_ids": [

655,

424,

401,

298,

... (more results will appear)

]

}

How it works:

Behind the scenes this endpoint is a wrapper on top of the LCBO API:

The parameter is taken from the request and forwarded to the LCBO API by making the following request:

GET lcboapi.com/products?q="Smirnoff Vodka"

Which returns:

```
{
  "status": 200,
  "message": null,
  "pager": {
    "records_per_page": 20,
    "total_record_count": 27,
    "current_page_record_count": 20,
    "is_first_page": true,
    "is_final_page": false,
    "current_page": 1,
    "current_page_path": "/products?q=%22Smirnoff+Vodka%22
&page=1",
    "next_page": 2,
    "next_page_path": "/products?q=%22Smirnoff+Vodka%22&page=
2",
    "previous_page": null,
    "previous_page_path": null,
    "total_pages": 2,
    "total_pages_path": "/products?q=%22Smirnoff+Vodka%22
&page=2"
  },
  "result": [
    {
      "id": 38505,
      "is_dead": false,
      "name": "Smirnoff Vodka",
      "tags": "smirnoff vodka spirits canada region not specified
schenley diageo bottle",
      "is_discontinued": false,
      "price_in_cents": 6000,
      "regular_price_in_cents": 6000,
      "limited_time_offer_savings_in_cents": 0,
      "limited_time_offer_ends_on": null,
      "bonus_reward_miles": 0,
      "bonus_reward_miles_ends_on": null,
      "stock_type": "LCBO",
      "primary_category": "Spirits",
```

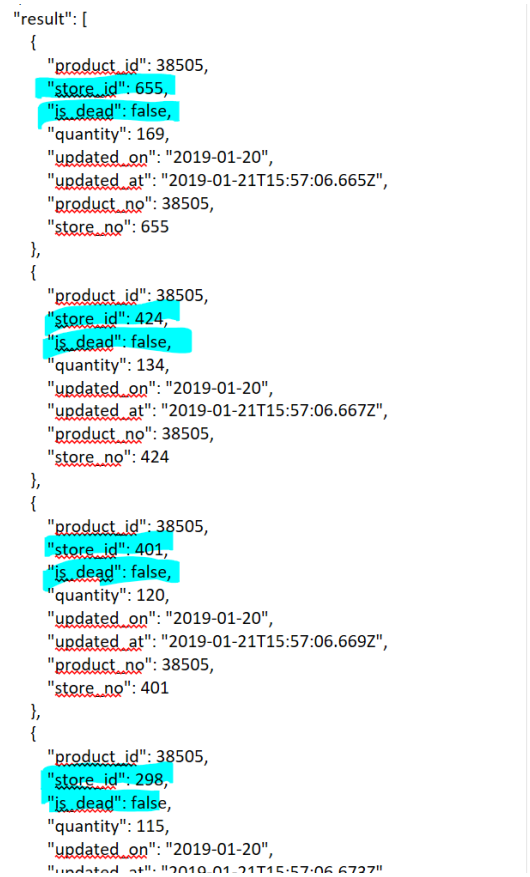

There is more to the request, but all we need is the LCBO product ID. This id is then used to make the following request to the LCBO API:

GET lcboapi.com/inventories?product_id=3805

Which returns:


```
{
  "status": 200,
  "message": null,
  "pager": {
    "records_per_page": 50,
    "total_record_count": 637,
    "current_page_record_count": 50,
    "is_first_page": true,
    "is_final_page": false,
```


… (skipping of irrelevant things in the request which can be made on your own by replacing lcboapi.com with the CIS 4450 Server IP:Port – host: 131.104.49.90:3000)…



The request returns 637 store ids (that are not dead/closed), these are just a couple for demonstration sake. The store ids (that are not dead/closed) are then returned to original caller who made the PyDrink.com/find/stores request.

GET PyDrink.com/find/location

This request takes the optional parameter store_id, if it is provided, the URI would like like this:

GET PyDrink.com/find/location?store_id=”655”

If not provided, all LCBO store locations will be returned otherwise, the following is the response for the above request (Status Code 200):

{

"name": "Main & Bay Street",

"address_line_1": "35 First Street",

"address_line_2": "P.O. Box 310",

"city": "Moosonee",

"postal_code": "P0L1Y0",

"telephone": "(705) 336-2301"

}

How it works:

Behind the scenes this endpoint is a wrapper on top of the LCBO API:

The parameter is taken from the request and forwarded to the LCBO API by making the following request:

GET lcboapi.com/stores/655

Which returns:


```
"status": 200,
"message": null,
"result": {
  "id": 655,
  "is_dead": false,
  "name": "Mississauga Rd & William Pkwy",
  "tags": "mississauga rd william pkwy 9445 brampton l6x0z8",
  "address_line_1": "9445 Mississauga Rd.,",
  "address_line_2": null,
  "city": "Brampton",
  "postal_code": "L6X0Z8",
  "telephone": "(905) 456-8145",
```


… (skipping of irrelevant things in the request which can be made on your own by replacing lcboapi.com with the CIS 4450 Server IP:Port – host: 131.104.49.90:3000)…

All we need is the highlighted text. If the store is not dead (closed) the name, address, city, postal code, and telephone is then compiled and returned the original caller who made the PyDrink.com/find/location request as the response

POST PyDrink.com/find

This request takes the drink_name as the body of the request:

Headers: None

Body (JSON):

```
{
        "drink_name": "Smirnoff Vodka"
}
```

And returns the following:

```
{
  "message": Found 637 LCBO stores that carry Smirnoff Vodka.
  "result": [
    {
```

```
        "name": "Mississauga Rd & William Pkwy",
        "address_line_1": "9445 Mississauga Rd.,",
        "city": "Brampton",
        "postal_code": "L6X0Z8",
        "telephone": "(905) 456-8145",
    },
    {
        "name": "Main & Bay Street",
        "address_line_1": "35 First Street",
        "address_line_2": "P.O. Box 310",
        "city": "Moosonee",
        "postal_code": "P0L1Y0",
        "telephone": "(705) 336-2301",
    },
    {
        "name": "Portage Rd & Colborne E",
        "address_line_1": "3714 Portage Road",
        "city": "Niagara Falls",
        "postal_code": "L2J2K9",
        "telephone": "(905) 356-3972",
    },
    ... (more results will appear)
  ]
}
```


As you can see this endpoint is the composition of the above two endpoints

The endpoint makes the stores GET request and then for every store_id's index makes a location request. It then aggregates this data and returns the resulting information as seen above.

The reason I split up one endpoint into 2 smaller endpoints and created another endpoint to compose the 2 together is to allow for modularity. If I want to search for only one specific store, I can still do that instead of being coupled with one endpoint which returns information on what could be (as shown with Smirnoff Vodka, a popular Vodka brand) a lot of stores and their location information.

The rest of the endpoints in the original table on page 1 and 2 follow a similar structure to the above 3 endpoints except they create/update/delete stores with their own inventory/catalogue of drinks or location information attached to a given store id.

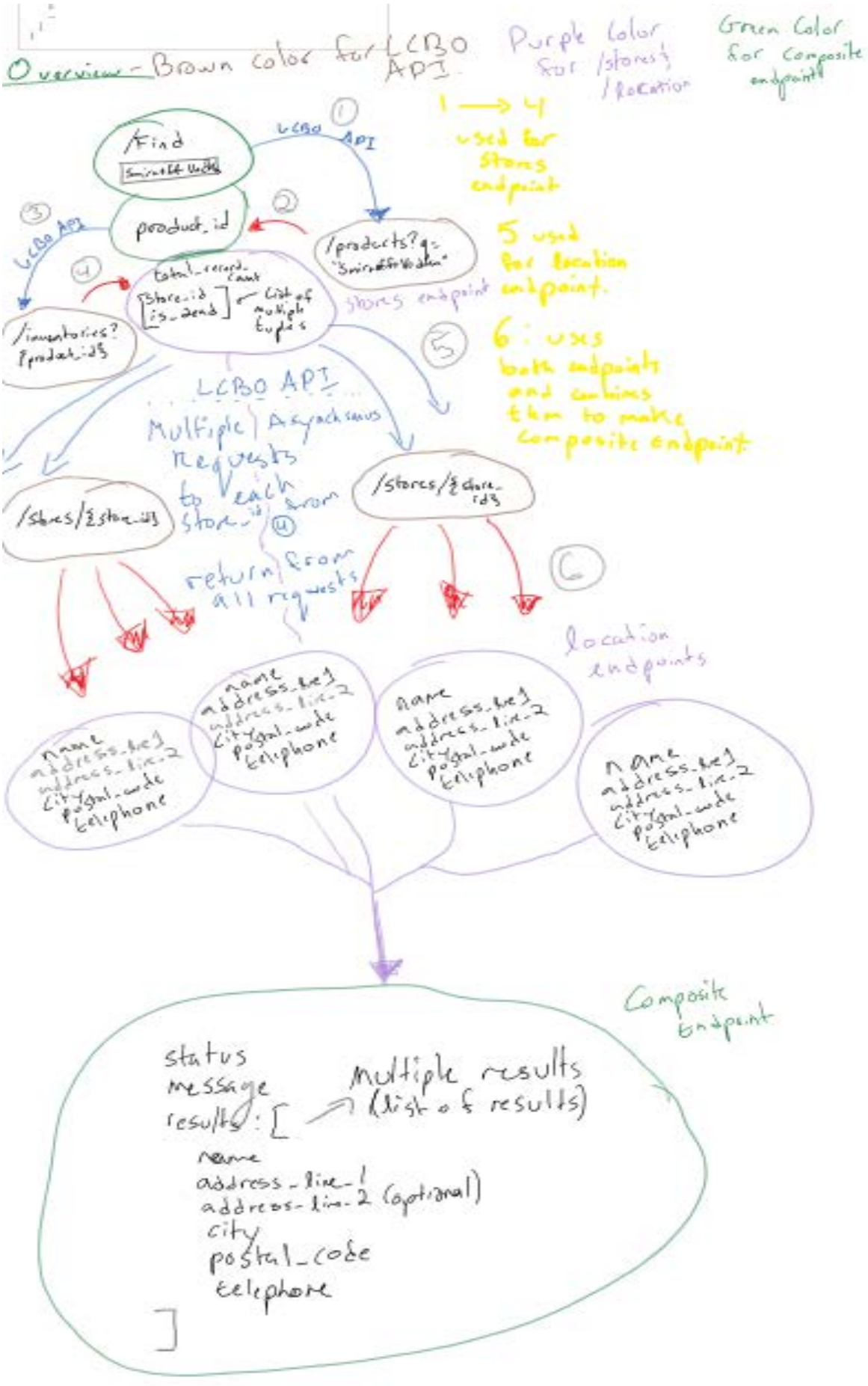
On the next page is a diagram that attempts to explain how the above 3 endpoints work from a more highly abstracted point of view.

Proposed Technologies

The technologies that can be used with this API is dependent on the developer using it. The only main requirements are those imposed by the REST protocol itself. It requires authentication which was not covered in this Design document but would most likely need another Endpoint which provided a username and password would return an Authorization Bearer Token that can be used for Admin restricted endpoints. The API also requires a running LCBO API and a configuration file that tells this API where the LCBO API is located (host and port). A server is needed to host the models from the LCBO API which run as docker containers (Redis, Postgres, and Ruby on Rails) as well as to host any authentication and session handling databases (Redis again). Since REST does not reinforce any specific rules, the technology options besides the server and LCBO API running alongside this API are relatively open to whomever is developing the system and using the API.

API and OO Preservation + Enhancement

My API allows transportation of information about Drink object's inventory information and gives room for a modular approach to finding information about a drink's availability without highly coupling a Drink to a large number of parameters that are required to find stores that carry it. This way it preserves the low coupling that I maintained between objects and enhances the OO design by requiring less Models to get the same results (no need to maintain an Inventory, Product, or Store model). Furthermore, this would help enhance cohesion since the data relevant to the program will relate to a single model (or maybe a couple if you decide to use ../find/stores and ../find/location separately) that will handle the methods of the API. The REST API only provides information that is relevant and hides information that has no relation to the methods and attributes of the model nor the rest of the application. Overall, the information hiding will make the design of the system less complex and increase code reuse since only one model is needed to maintain the API and can be used by other models to get information from the API; rather than every model contacting the API directly and doing the same operations. The API is also modular in the sense that not only the Composite Endpoint must be used, rather if an application desires to use this API they can access just store ids and interface with the LCBO API directly for different information or get location information on specific stores only rather than all the stores since that is more efficient.


Overview - Brown color for LCBO API
Purple color for /stores/{}/location
Green Color for Composite endpoint
1 → 4 used for stores endpoint
5 used for location endpoint.
6: uses both endpoints and combines them to make composite endpoint
Find
product_id
/products?q="Swinnerton-Dunn"
/inventories?product_ids
total_record_count
[store_id, is_dead] ← List of multiple tuples
stores endpoint
LCBO API
Multiple Asynchronous requests to each store_id from
return from all requests
/stores/{store_id}
/stores/{store_ids}
location endpoints
name
address_line_1
address_line_2
city
postal_code
telephone
Composite endpoint
status
message
results: [
Multiple results (list of results)
name
address_line_1
address_line_2 (optional)
city
postal_code
telephone
]