* Search a read-only book catalog.
* Create, update and display shopping carts.

Each one of these will be modelled as a **collection** of **collection items**.

Each collection will be available behind a RESTful **collection URL**. A collection URL COLL\_URL can typically be set up as follows:

* A POST to COLL\_URL creates a new item in the collection with a 201 CREATED empty response. The Location header in the response should specify the **item-URL** of the newly created item which is typically along the lines of COLL\_URL/ID, where ID is an ID for the newly created item.

Note that this is not supported for the books collection since the books catalog is read-only.

* A GET to COLL\_URL enhanced with query-parameters will search for all items in the collection which satisfy the query parameters. The response will usually contain next/previous links to facilitate scrolling through the results.

Note that the carts collection does not support search.

* A GET to an *item-URL* will return the details of that collection item.
* A PATCH to an *item-URL* will update that collection item with parameters specified in the request body with a 204 No Content HTTP status.

Note that this is not supported for books since they are read-only.

Note that a request to an invalid item URL should result in a 404 Not Found HTTP response as usual.

**Required**

./index.mjs PORT MONGO\_DB\_URaL [DATA\_FILE...]

will start a web server listening on PORT backed up by the database specified by MONGO\_DB\_URL.

The DATA\_FILE arguments should specify \*.json or \*.json.gz files containing book data. If there is at least one DATA\_FILE argument, then the database is cleared of **all** data (both books and carts).

The responses of the web services you build can be of two types:

* An **error response**: The response must be a JSON object with two properties:

*status*

This must be an integer and must match the HTTP status for the response.

*errors*

This must be a list of errors, where each error must be a JSON object containing the following properties:

*code*

A string giving a code for the error.

*message*

The error message which should be as detailed as possible.

*name*

The internal name of the widget which is the proximate cause of the error (optional).

Note that this must be the format for **all** error responses.

* A **success response**: All non-empty success response must be a JSON object, possibly containing the following two properties:

*result*

The main result of the web service. Documented for each web services.

*links*

This must be a list of links for the response, where each link must be a JSON object containing the following properties:

*href*

An absolute URL for the linked resource.

*rel*

The relationship between the response and the linked resource.

*name*

A description for the link.

This top-level links property must always contain a *self-link* with rel and name both set to self and href set to the URL which generated the response.

What you specifically need to do is add code to the provided ws-server.mjs source file to implement the following URLs served on http://localhost:PORT:

*GET /api*

There is no result property, but the links property must contain the following 3 links:

* A *self-link* as documented above.
* A *books-link* with rel set to collection and name set to books and href set to a collection URL for books. The value for href is entirely up to you as long as it is subordinate to /api; the sequel refers to this href value as the *books-collection-url*.
* A *carts-link* with rel set to collection and name set to cart and href set to a collection URL for carts. The value for href is entirely up to you as long as it is subordinate to /api; the sequel refers to this href value as the *carts-collection-URL*.

*GET books-collection-URL*

Conduct a search for books which match specified query parameters isbn, authorsTitleSearch, \_index and \_count. Return result as a list of matching results sorted by book title. The result list can contain up to \_count results (default 5), starting at \_index (default 0). The result list should be empty if there are no matching results.

Each individual book item in the result list should be enhanced with a links property containing a single link with rel set to details, name set to book and href set to the *book-item-URL* for that book item.

The overall response should also have a top-level links property containing up to 3 links:

* A *self-link* as described above. This must always be present and must point to the URL resulting in this response.
* A *next-link* with rel and name both set to next and href set to a URL which can be used to get the next set of \_count results. This link should be present only if there are subsequent results for the search.
* A *previous-link* with rel and name both set to prev and href set to a URL which can be used to get the previous set of \_count results. This link should be present only if there are previous results for the search.

The last two links can be used for scrolling through search results.

*GET book-item-URL*

The result property is set to the book identified by *book-item-URL*. The links property should be a list containing a single *self* link. The response should fail with a 404 error if there is no such book.

*POST carts-collection-URL*

This should create a new shopping cart. The response should be empty with status 204 CREATED with a Location header set to the URL of the newly created cart. In the sequel, the URL for an individual cart is referred to as the *cart-URL*.

*GET cart-URL*

Display the contents of the specified cart. A successful response should have the following properties:

*'\_lastModified'*

A timestamp giving the time the cart was last modified.

*links*

A list containing a single *self-link*.

*result*

A list of the cart items. Each item should have sku and nUnits properties as well as a links property specifying a single link with rel set to item, name set to book and href set to the *book-item-URL* for the book item corresponding to the sku.

*PATCH cart-URL*

This request must have a JSON body giving the sku and nUnits to be updated. The cart specified by *cart-URL* is updated with the specified sku and nUnits with a return HTTP status of 204 No Content. If the sku does not correspond to the ISBN of a book in the catalog, then the HTTP response should be a 404 Not Found.

**Provided Files**

It contains the following files:

*ws-server.mjs*

A skeleton file. You should be doing all your development in this file.

*index.mjs*

The file invoked on the command-line. It is a trivial wrapper which simply calls cli.mjs.

*cli.mjs*

This file provides the complete command-line behavior which is required by your program. It requires model.mjs. You **must not** modify this file; this ensures that your server meets its command-line specifications.

*model.mjs*

This file has been provided only to ensure that the code in the directory can be run out-of-the-box.

*meta.mjs*

Meta-information about the different model object categories.

*model-error.mjs*

A trival class for application errors.

*validator.mjs*

Validation code for checking for local errors which depend only on a single object instance. Note that it provides generic validation based on types for input parameters. The model will need to perform additional validation for checking for global errors across objects.