

Web Services GET exercise (Java)

In this exercise, you'll work on a command-line application that displays online auction info. A portion of the command-line application is provided. You'll write the remaining functionality.

You'll add web API calls using RestTemplate to retrieve a list of auctions, details for a single auction, and filter the list of auctions by title and current bid.

These are the endpoints you'll work on for this exercise:

- GET: `http://localhost:3000/auctions`
- GET: `http://localhost:3000/auctions/{id}`
- GET: `http://localhost:3000/auctions?title_like=<value>`
- GET: `http://localhost:3000/auctions?currentBid_lte=<value>`

Step One: Start the server

Before starting, make sure the web API is up and running. Open the command line and navigate to the `./server/` folder in this exercise.

First, run the command `npm install` to install any dependencies. You won't need to do this on any subsequent run.

To start the server, run the command `npm start`. If there aren't any errors, you'll see the following, which means that you've successfully set up your web API:

```
\{^_^}/ hi!  
  
Loading data-generation.js  
Done  
  
Resources  
http://localhost:3000/auctions  
  
Home  
http://localhost:3000  
  
Type s + enter at any time to create a snapshot of the database  
Watching...
```

Step Two: Explore the API

Before moving on to the next step, explore the web API using Postman. You can access the following endpoints:

- GET: `http://localhost:3000/auctions`
- GET: `http://localhost:3000/auctions/{id}` (use a number between 1 and 7 in place of `{id}`)

Step Three: Review the starting code

Data model

There's a class provided in `/src/main/java/com/techelevator/auction/Auction.java` that represents the data model for an auction object. If you've looked at the JSON results from the API, the properties for the class should look familiar.

Provided code

In `App.java`, you'll find three methods that print information to the console:

- `printGreeting()`: Prints menu options and routes to methods for each option
- `printAuctions()`: Prints a list of auctions
- `printAuction()`: Prints a single auction

Each of these methods is called in response to the menu selection within the `while` loop. The `if` statements determine which method to call based upon the menu selection. Notice that each of the four methods that you'll implement are called there as well.

Your code

There are four other methods where you'll add code to call the API methods:

- `listAllAuctions()`
- `listDetailsForAuction()`
- `findAuctionsSearchTitle()`
- `findAuctionsSearchPrice()`

In the `run()` method, find the `while` loop. Notice how each menu option is evaluated and these methods are called. The `while` loop returns users back to the menu.

Step Four: Write the console application

There are three variables declared for you at the top of the class that you can use throughout your exercise:

```
public class App {  
  
    public static final String API_URL = "http://localhost:3000/auctions";  
    public static RestTemplate restTemplate = new RestTemplate();  
    private static Scanner scanner;  
  
    // ...  
  
}
```

List all auctions

In the `listAllAuctions()` method, find the comment `//api code here`. Add code here to:

- Use the RestTemplate to request all auctions and save them into an array of Auctions
- Replace the current return statement to return the array of auctions

Once you've done this, run the unit tests. After the tests `listAllAuctions()` and `listAllAuctionsNoAuctions()` pass, you can run the application. If you select option 1 on the menu, you'll see the ID, title, and current bid for each auction.

List details for a specific auction

In the `listDetailsForAuction()` method, find the `//api code here` comment. Add code here to:

- Use RestTemplate to request a specific auction by ID
- Return the single auction

Once you've done this, run the unit tests. After the test `listDetailsForAuction()` passes, you can run the application. If you select option 2 on the menu, and enter an ID of one of the auctions, you'll see the full details for that auction.

Find auctions with a specified term in the title

In the `getByTitle(String title)` method, find the `//api code here` comment. Add code here to:

- Use RestTemplate to request all auctions which match the `title` parameter and save them into an array of Auctions.
- The API URL requires a query string with the key named `title_like` and the value of the `title` parameter. This requests the server to search for auctions that have a title containing the value in the `title` parameter.
- Replace the current return statement to return the array of auctions. Return null if no auctions were found.

Once you've done this, run the unit tests. After the tests `findAuctionsSearchTitle()` and `findAuctionsSearchTitleNotFound()` pass, run the application. If you select option 3 on the menu, and enter a string, like `watch`, you'll see the ID, title, and current bid for each auction that matches.

Find auctions below a specified price

The final method, `getByPrice(double price)`, also uses a query string. The parameter key is `currentBid_lte` and has the value of the `price` parameter. This requests the server to search for auctions that have a `currentBid` that's **Less Than or Equal** to the value you supply.

Find the `//api code here` comment and add code here to:

- Use RestTemplate to request all auctions which are less than or equal to the `price` parameter and save them into an array of Auctions.
- The API URL requires a query string with the key named `currentBid_lte` and the value of the `price` parameter. This requests the server to search for auctions that have a price less than or equal to the value in the `title` parameter.
- Replace the current return statement to return the array of auctions. Return null if no auctions were found.

Once you've done this, run the unit tests. After the tests `findAuctionsSearchPrice()` and `findAuctionsSearchPriceNotFound()` pass, run the application. If you select option 4 on the menu and enter a number, like `150`, you'll see the ID, title, and current bid for each auction that matches.

Since the value is a `double`, you can enter a decimal value, too. Try entering `125.25`, and then `125.20`, and observe the differences between the two result sets. The "Mad-dog Sneakers" don't appear in the second list because the current bid for them is `125.23`.