Server-Side APIs: Part 2 Exercise (Java)

In this exercise, you'll add on to the auctions application you previously worked on. When you first built out the application, you added the ability to list, get, and search auctions by title and current bid. In this exercise, you'll add the ability to update and delete auctions. You'll also need to perform data validation to inform the client of any problems.

Step One: Import project into IntelliJ and explore starting code

Import the server-side APIs part 2 exercise into IntelliJ. After you've imported the project, review the starting code. The code likely looks familiar to you as it's a continuation of the previous exercise.

`ResponseStatusException`

The `get()` method in the `AuctionController` now throws a `ResponseStatusException` if the DAO returns `null`, which means there's no auction for the id provided. It's best practice to return a `404(NotFound)` status code when a resource isn't found:

```
@RequestMapping(path = "/{id}", method = RequestMethod.GET)
public Auction get(@PathVariable int id) {
    Auction auction = dao.get(id);
    if (auction == null) {
      throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Auction Not
Found");
    } else {
      return dao.get(id);
    }
}
```

Use `ResponseStatusException` and `@ResponseStatus` to specify the status code returned from your methods.

Tests

The `src/test/java/com/techelevator/auctions/controller` package contains the `AuctionsControllerIntTest` class. It contains the tests for the methods you'll write for this exercise. More tests pass after you complete each step.

In `src/test/java/com/techelevator/auctions/model/AuctionValidationTest`, you'll find a new set of unit tests. These tests verify that you're validating incoming data.

Feel free to run the server and test the application in the browser, or in Postman. However, your goal is to make sure all the tests pass.

Step Two: Modify the `create()` method

First, work on the `create()` method. When you create a new auction, send a status code of `201(Created)` back to the client. After you complete this step, the `create_ValidAuction_ShouldAddNewAuction` test passes.

Step Three: Add auction data validation

Right now, you can send in an object with a blank title, description, and user. Because there's no data validation, the system creates one. You'll need to add these rules to `Auction.Java`:

- title
  - rule: Not Blank
  - message: "The title field must not be blank."
- description
  - rule: Not Blank
  - message: "The description field must not be blank."
- user
  - rule: Not Blank
  - message: "The user field must not be blank."
- currentBid
  - rule: Min 1
  - message: "The currentBid field must be greater than 0."
  -
    - currentBid is a double, so you can't use the rule `@Min()`.
  -
    - `@Positive` might be an annotation to look at.

Afterwards, run the unit tests in `src/test/java/com/techelevator/auctions/model/AuctionValidationTest.java` to verify that you have the correct validations in place.

Step Four: Update the controller's `create()` method

To enforce validation in the controller, add an annotation before the `Auction` argument in the `create()` method to tell Spring to validate the object. If completed properly, the `create_InvalidAuction_ShouldNotBeCreated` test passes.

Step Five: Implement the `update()` method

This method updates a specific auction. Passed into the method as a parameter is the updated auction.

In `AuctionController.Java`, create a method named `update()` that accepts an `Auction`, the auction's ID, and returns the updated `Auction`. Then add the `@RequestMapping` annotation to this method so it responds to `PUT` requests for `/auctions` with a number following it, like `/auctions/7`. Next, pass a value to the path to tell it to accept a dynamic parameter.

This method must also:

- Return an `Auction` from `dao.update()`, passing to it the auction and ID parameters.
- Respond to the client with the appropriate status when the ID isn't in the datastore.
  - Like the `dao.get()` method, the `dao.update()` method returns null when the ID isn't valid.

After you complete this step, the `update_ValidAuction_ShouldUpdateExistingAuction`, `update_InvalidAuctionShouldNotBeUpdated` and `update_InvalidAuctionId_ShouldReturnNotFound` tests pass.

Step Six: Implement the `delete()` method

This method deletes a specific auction.

In `AuctionController.Java`, create a method named `delete()` that accepts an `int` and returns `void`. Then add the `@RequestMapping` annotation to the method so it responds to `DELETE` requests for `/auctions` with a number following it. Next, pass a value to the path to tell it to accept a dynamic parameter.

This method must also:

- Call `dao.delete()`, passing to it the ID parameter.
- Respond with a `204(No Content)` status code back to the client, as the method doesn't return a value.

If completed properly, the `delete_ShouldReturnNoContent` test passes.

If you followed the instructions correctly, all tests now pass.