WA2985 Booz Allen Hamilton MSD Program - Spring and Gradle - Week 1

Student Labs

LabGuide 2

(Spring Project)

Web Age Solutions Inc.

Table of Contents

Lab 1 - Spring Project – Create the RESTful Stock Account Service

In this lab, you will create and test a simple RESTful Web Service API to retrieve stock accounts. You will use Postman, a popular REST API tool, for unit testing the web service.

Part 1 - Stock Account API Requirements

The Stock Account API must allow users to:

- Retrieve a list of stock accounts
- Retrieve a stock account by id
- Retrieve a stock account by name
    - If there's more than one stock account holder with the same name, then return the first matching account.
    - If there's no matching account, then return null.
    - Do not return a collection of stock accounts. Only return back a single stock account.

The endpoints must return the stock accounts in JSON format.

Part 2 - Examine the Starter Application

In this part, you will import a starter application and examine the existing code.

__1. Select **File → Import** from the menu bar.

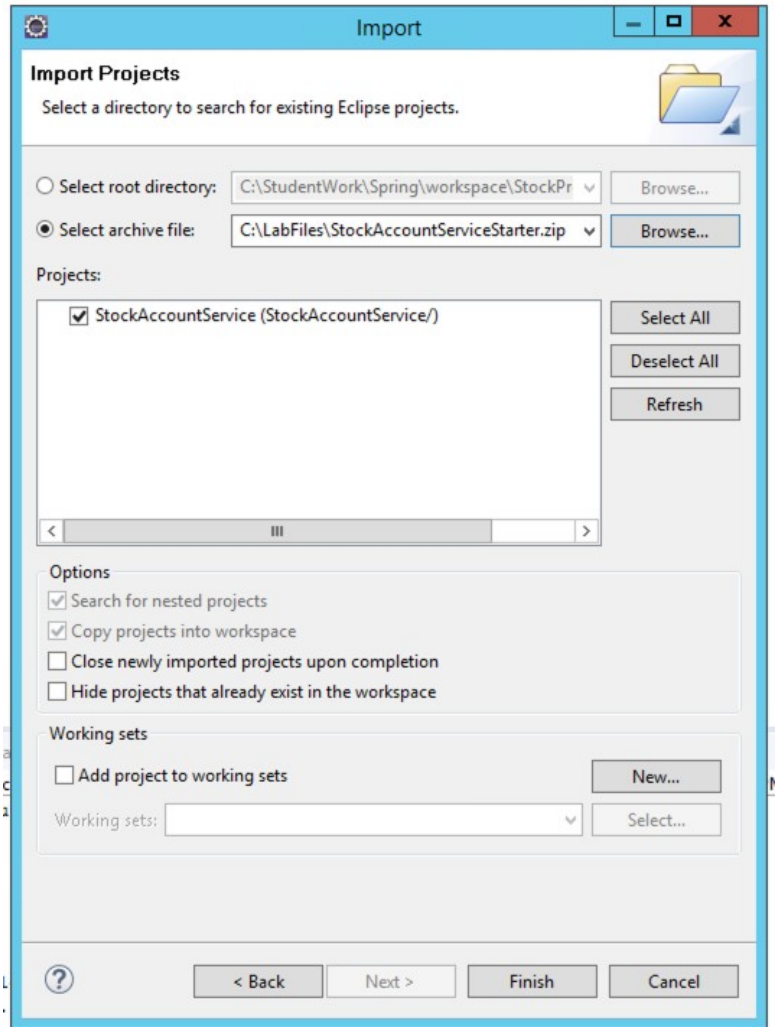
__2. Expand **General** and select **Existing Projects into Workspace**.

__3. Click **Next**.

__4. Click the **Select archive file**.

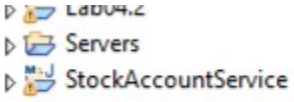
__5. Click the **Browse** button and navigate to **C:\LabFiles\StockAccountServiceStarter**.

__6. Click **Open**.



__7. Click **Finish**.

__8. Confirm the project is imported and there are no errors. There will be some warnings.



__9. Open the **pom.xml** file and scroll to the bottom of the file.

__10. The **finalName** element contains the value **stockservice**. This corresponds with the context root (path) of the web application.

```
50
51⊖    <build>
52         <finalName>stockservice</finalName>
```

__11. Examine the classes in the **com.stock.domain** package.

__12. Examine the classes in the **com.stock.persistence** package.

__13. Examine the classes in the **com.stock.service** package.

__14. Finally, examine the classes in the **com.stock.config** package. There are several **TODO** items that you will complete in the next part.

Part 3 - Complete the Spring Configuration

In this part, you will complete the Spring configuration for the project.

__1. Expand the **com.stock.config** package.

__2. Open the **StockWebApplicationInitializer.java** file.

__3. Complete the **TODO** task to fill in a URI of **/api/*** for the DispatcherServlet.

Note: Unlike in the labs, we're using a URI of /api/*, instead of /rest/*, for the DispatcherServlet. This is another commonly used URI for RESTful APIs.

__4. Open the **SpringConfig.java** file.

__5. Complete the **TODO** tasks to declare the class as a configuration class and import the SpringRepositoryConfig and SpringServicesConfig classes.

__6. Open the **WebConfig.java** file.

__7. Complete the **TODO** tasks to declare the class as a configuration class, add the annotation to enable Spring MVC and add the annotation to specify auto-scan of com.stock.

__8. Open the **SpringRepositoryConfig.java** file.

__9. Complete the **TODO** tasks to declare the class as a configuration class and declare the stock account repository bean.

__10. Open the **SpringServicesConfig.java** file.

__11. Complete the **TODO** tasks to declare the class as a configuration class, inject a repository and declare the stock account service bean.

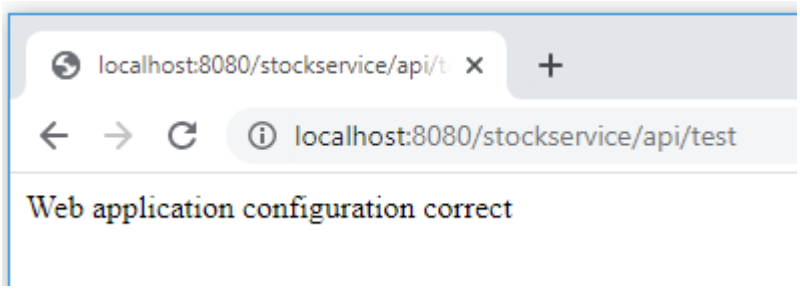
Part 4 - Test the Configuration

In this part, you will test the web configuration and the root Spring configuration.

__1. Open Chrome.

__2. Access the following URL: **http://localhost:8080/stockservice/api/test**

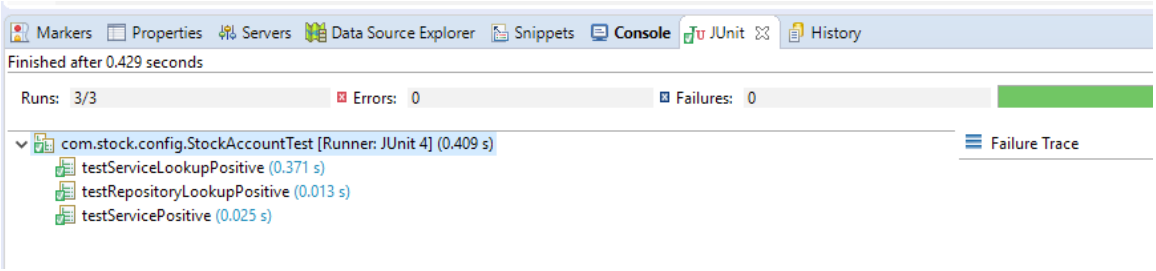
__3. Verify you see the following page:



If you don't see the above page, go back to Part 3 and verify your web configuration.

__4. Run the unit tests contained inside **com.stock.config.StockAccountTest**.

__5. Verify all three tests pass.



If they don't, inspect the **JUnit Failure Trace** and then go back to Part 3 and verify your root Spring configuration.

Part 5 - Configure the Stock Account Resource

In this part, you will configure the stock account resource, which will serve as the implementation of our RESTful stock account service.

__1. Expand the **com.stock.rest** package.

__2. Open the **StockAccountResource.java** file.

__3. Complete the **TODO** tasks to declare the class is a REST controller and to map the class to the **/accounts** URI.

__4. Complete the **TODO** task to inject the stock account service.

Part 6 - Implement the retrieval of a list of stock accounts

In this part, you will fulfill the first API requirement by implementing and testing the web service endpoint which retrieves a list of stock accounts. You will use Postman to unit test the endpoint.

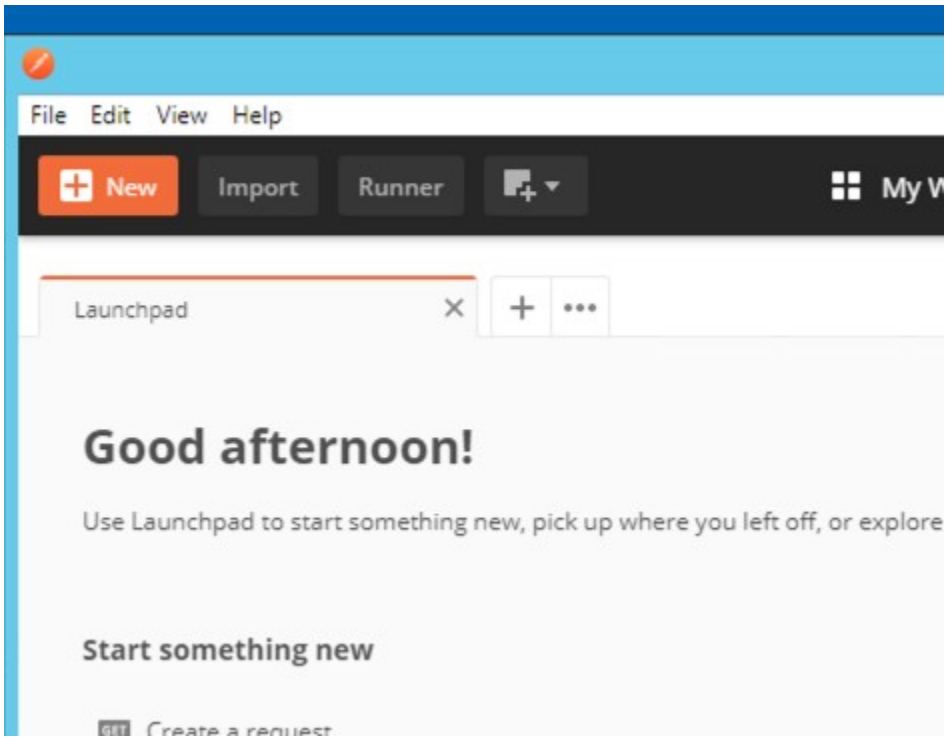
__1. In the **StockAccountResource.java** class, complete the **TODO** task to create a method to retrieve all the stock accounts.

__2. Add the project to the server (i.e., Tomcat).

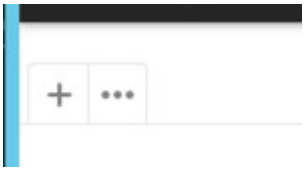
__3. Start the server.

__4. Open Postman by double clicking on the **Postman** shortcut on your desktop.

__5. Close the **Launchpad**.



__6. Click the + button next to the ellipsis **(…)** button.



__7. Make sure **GET** is selected from the drop-down, which corresponds with the HTTP method.

__8. Type your URL into the location bar. Remember the URL has the following syntax:

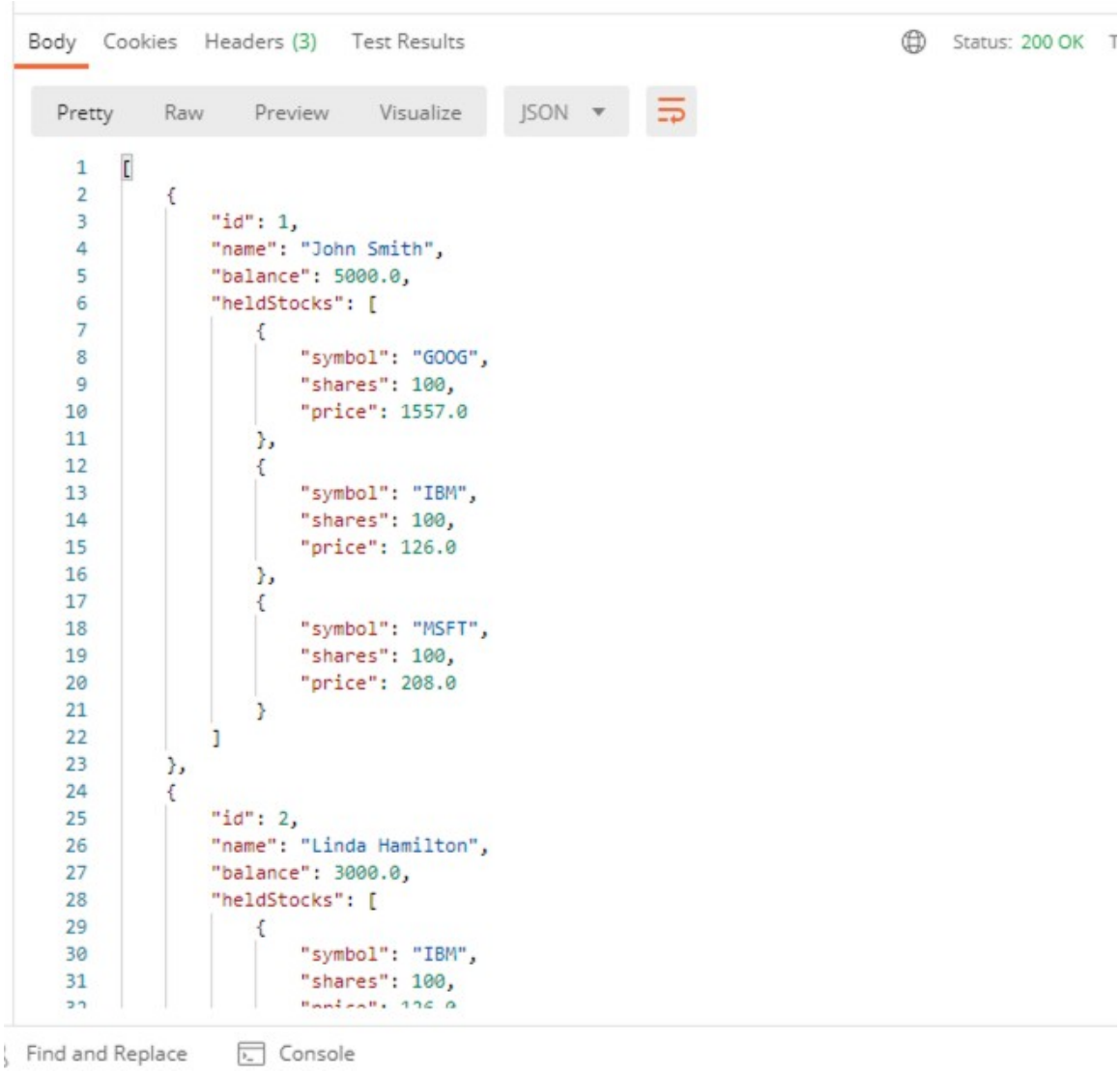
```
http://<hostname>:<port>/<WebApplicationContextRoot>/
<DispatcherServletURI>/<RestControllerURI>/<RestControllerMethodURI>
```

__9. Click the blue **Send** button.

__10. Verify you get back a list of 5 accounts.

Note: The number of stocks for each account holder will vary, since it's randomized.

11. Verify you get back an HTTP status code of **200**. The status code is displayed in green. This means the request was successfully processed by the server.



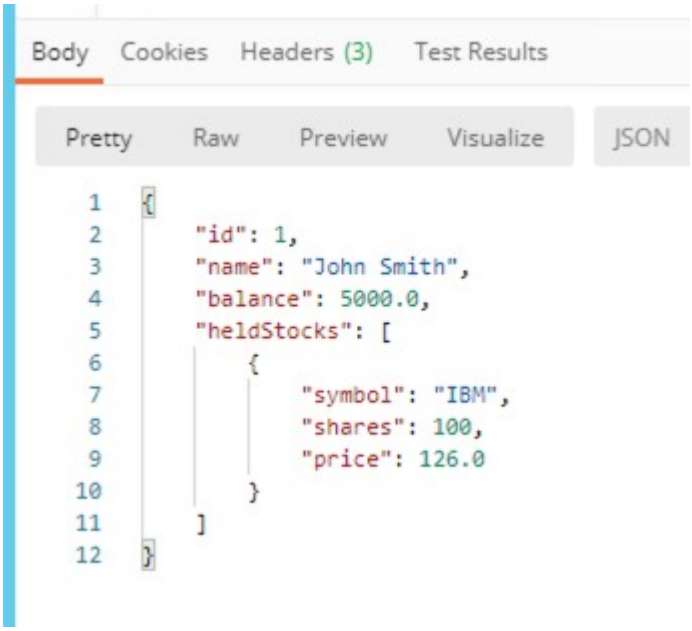
Part 7 - Implement the retrieval of a stock account by id

In this part, you will fulfill the second API requirement by implementing and testing the web service endpoint which retrieves an individual stock account by id.

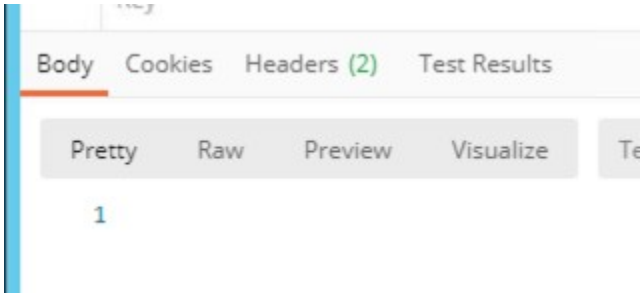
__1. In the **StockAccountResource.java** class, complete the **TODO** task to create a method to retrieve a stock account by id.

__2. Deploy the changes to the server by restarting the server.

__3. Using Postman, test the web service endpoint. First test a valid account id.



__4. Now test an invalid account id. You should get back an empty response, since there is no account.



__5. Verify the server still returns a **200** HTTP status code.

Ideally it should return a **404** (i.e., NOT FOUND). Later, when we cover Spring Boot, we'll learn how to specify the HTTP status code returned. For now, we'll simply return the default value of 200.

__6. Try supplying a non-numeric value for the stock account id, such as "one".

__7. Verify you get back an HTTP status code of **400** (i.e., NOT FOUND). This means the client provided an invalid request to the server. The server was expecting a number

for the id and you didn't supply one.

Part 8 - Implement the retrieval of a stock account by name

In this part, you will fulfill the third and final API requirement by implementing and testing the web service endpoint which retrieves an individual stock account by name.

__1. In the **StockAccountResource.java** class, complete the **TODO** task to create a method to retrieve a stock account by name.

Note that the stock account service returns a collection of stock accounts which match the name. Return the first one that's matched. If there's no matching account, then return null.

Do not return a collection of stock accounts. Only return back a single stock account.

Note: Remember that the URIs defined inside your resource class must be unique. Otherwise, you will get an error indicating there are ambiguous handler methods defined in your resource class.

__2. Deploy the changes to the server by restarting the server.

__3. Using Postman, test the web service endpoint. Test "John Smith".

__4. There are two account holders with the name "John Smith". Verify you get back only the first one. Also verify you only get back a single object and not an array of objects.

__5. Verify the server returns a **200** HTTP status code.

```
{
    "id": 1,
    "name": "John Smith",
    "balance": 5000.0,
    "heldStocks": [
        {
            "symbol": "GOOG",
            "shares": 100,
            "price": 1557.0
        },
        {
            "symbol": "IBM",
            "shares": 100,
            "price": 126.0
        }
    ]
}
```


__6. Now test an invalid account name. You should get back an empty response, since there is no account.

__7. Verify the server returns a **200** HTTP status code.

Ideally it should return a **404** (i.e., NOT FOUND). Once again, we'll learn how to accomplish this later.

Part 9 - Review

In this project, you created a RESTful Stock Account Service and tested the operations using Postman. Congratulations!