Project 4 - DAO Pattern and REST API

15/20 Points

2024/6/19

Attempt 1

Review Feedback
2024/6/19

Attempt 1 Score:
15/20

View Feedback

Anonymous Grading: **no**

Unlimited Attempts Allowed

Details

Points: 20 points
Deliverables: Upload [Name]BookstoreRest.war to Archive Upload (http://cs5244.cs.vt.edu:8080/ArchiveUpload/) and submit the URL to Canvas.
Resources:

- dao-business.zip (https://drive.google.com/file/d/1v8c8tUGJMz5nnCZDuYD5oW1dzALKfip/view?usp=sharing),
- rest-api.zip (https://drive.google.com/file/d/1xyX1gzWG8J7jZsqS6pb7WgvNhtUxkj1W/view?usp=sharing)

Part A: Setting up your database

Step 0 – Install Virginia Tech's Remote Access VPN

In order to access your database on the CS5244 server, you will have to install Virginia Tech's Remote Access VPN. Instructions on how to do that can be found at: https://www.nis.vt.edu/ServicePortfolio/Network/RemoteAccess-VPN.html (https://www.nis.vt.edu/ServicePortfolio/Network/RemoteAccess-VPN.html). You will need to connect to the VPN when you are trying to access your remote database.

Once you install it, connect to "VT Traffic over SSL VPN". You will only need to do this while working on your projects, because you need to access your database on the CS5244 server. You may have to quit any other VPNs while you are connect, but I'm not 100% sure on this. Let us know on Piazza if you run into any problems.

Step 1 – Create a new JakartaEE project in IntelliJ IDEA

Create a new project in IntelliJ IDEA. For reference, see "Setting up the Server Project (https://canvas.vt.edu/courses/192345/assignments/2099642)" from Module 3's mini project. Choose JakartaEE. For ArtifactId (project and WAR name) put [Name]BookstoreRest, where [Name] is your name according to Piazza post "Project Names" that is pinned to the top of your feed. Choose language: Java and Build system: Gradle, and JDK 21. Once the project has been created, open the build.gradle file and replace it with the build.gradle code below.

Assignment Project Quiz Exam Essay Help
WeChat: cestbon688
Email: accoder@163.com
overseas

Notice that we have added a several things to the gradle file. The dependencies will ensure that your index.jsp file runs correctly. . The index.jsp file will display a table containing information from your database. It will come up automatically when you run your project.

```
plugins {
    id 'java'
    id 'war'
}


repositories {
    mavenCentral()
}

ext {
    junitVersion = '5.10.0'
}

tasks.withType(JavaCompile).configureEach {
    options.encoding = 'UTF-8'
}

dependencies {

    compileOnly('jakarta.json.bind:jakarta.json.bind-api:3.0.0')
    compileOnly('jakarta.json:jakarta.json-api:2.1.1')

    compileOnly('jakarta.servlet:jakarta.servlet-api:6.0.0')
    implementation('org.glassfish.jersey.containers:jersey-container-servlet:3.1.3')
    implementation('org.glassfish.jersey.media:jersey-media-json-jackson:3.1.3')
    implementation('org.glassfish.jersey.inject:jersey-hk2:3.1.3')
    implementation('mysql:mysql-connector-java:8.0.33')
    testImplementation("org.junit.jupiter:junit-jupiter-api:${junitVersion}")
    testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:${junitVersion}")


    // For JSON serialization/deserialization
    implementation('com.fasterxml.jackson.core:jackson-core:2.15.1')
    implementation('com.fasterxml.jackson.core:jackson-databind:2.15.1')
    implementation('com.fasterxml.jackson.core:jackson-annotations:2.15.1')
    implementation('com.fasterxml.jackson.module:jackson-module-jaxb-annotations:2.15.1')
    implementation('com.fasterxml.jackson.datatype:jackson-datatype-jsr310:2.15.1')


    // For viewing the database using a JSP
    implementation('jakarta.servlet.jsp.jstl:jakarta.servlet.jsp.jstl-api:3.0.0')
    implementation('org.glassfish.web:jakarta.servlet.jsp.jstl:3.0.1')


    // JAX-B dependencies for JDK 9+
    implementation('javax.xml.bind:jaxb-api:2.3.1')
    implementation('com.sun.xml.bind:jaxb-core:2.2.11')
    implementation('com.sun.xml.bind:jaxb-impl:2.2.11')
    implementation('javax.activation:activation:1.1.1')
}

test {
    useJUnitPlatform()
}
```

Assignment Project Quiz Exam Essay Help
WeChat: cestbon688 Email: accoder@163.com

Step 2 – Setup a run configuration for your local Tomcat server

Go to the down arrow near the top right of IntelliJ. It says "Edit configurations..." when you click it. Click the [+] button at the top left and choose Tomcat -> Local. The dialog box that appears should

already have most of the information in it since you created a local Tomcat configuration for a previous project. Name the configuration "Tomcat". The bottom of the dialog box should give a warning saying that no artifacts are marked for deployment. Click [Fix it] using [Name]BookstoreRest.war and give the application context the name /[Name]BookstoreRest.

Run the project to make sure the default Hello index.jsp page comes up.

Step 3 – Setup the Database View

Go to View -> Tool Windows -> Database. Choose [+] -> Data Source -> MySQL. Fill out the information on the form:

- Host: cs5244.cs.vt.edu
- User: [pid]
- Password: [last 4 digits of 9 digit ID]
- Database: [Name]BookstoreDB

Test the connection to ensure that it works.

Step 4 – Setup your Database

Create a data.sql file in your project under src/main/resources (you may have to create the resources directory) that inserts category and book data for your bookstore. The beginning of the file should look something like this:

```
DELETE FROM book;
ALTER TABLE book AUTO_INCREMENT = 1001;

DELETE FROM category;
ALTER TABLE category AUTO_INCREMENT = 1001;

INSERT INTO `category` (`name`) VALUES ('Classics'),('Fantasy'),('Mystery'),('Romance');

INSERT INTO `book` (title, author, description, price, rating, is_public, is_featured, category_id) VAL
UES ('The Iliad', 'Homer', '', 699, 0, TRUE, FALSE, 1001);
INSERT INTO `book` (title, author, description, price, rating, is_public, is_featured, category_id) VAL
UES ('The Brothers Karamazov', 'Fyodor Dostoyevski', '', 799, 0, TRUE, FALSE, 1001);
INSERT INTO `book` (title, author, description, price, rating, is_public, is_featured, category_id) VAL
UES ('Little Dorrit', 'Charles Dickens', '', 599, 0, TRUE, FALSE, 1001);
...
```

Notice the the data in the tables are removed in reverse order to prevent foreign key constraint errors. Also, the auto-increment function for primary keys is reset so that numbers begin at 1001. Starting the numbers at 1001 ensures that we can inject test data into the database with primary keys that are under 1001. This lets us check how your application handles specific data. Note that we can only test your database if you DO NOT change the database schema that we gave you, so please keep the same table names, field names, types, and constraints that have been given to you.

I've thrown in a couple extra fields in the book table that are optional in the client, but you must implement them in the server: `description`, `rating`, and `is_featured`. Based on past courses, some students wanted to have the flexibility to add a short description of the book or even a star rating. Rather than allowing students to change the database structure (which complicates grading) I have

Assignment Project Quiz Exam Essay Help
Wechat: custbon_688
Email: accoder_overseas@163.com

added these optional fields. You do not need to use them in your client, but they *do* have to work in Project 4. I have included `is_featured` mainly because I'm going to use it in my book store. In every category, I'm going to feature one book simply by changing the color and background color of the book box.

Once you have your file, **copy and paste the code into the console and execute it.** View your book and category tables (double click the tables in the database view) to ensure that your data was added to the database tables correctly.

Step 5 – Setup context.xml

Your context.xml file is a Tomcat-specific deployment descriptor. It resides in the META-INF directory under webapp directory. Therefore, you must first create that directory. Once it is created, create a file named context.xml under that directory. Place the following code Inside the file:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/">
    <Resource name="jdbc/[Name]Bookstore"
              auth="Container"
              type="javax.sql.DataSource"
              maxTotal="2"
              maxIdle="0"
              maxWaitMillis="10000"
              username="[pid]"
              password="[pin]"
              driverClassName="com.mysql.cj.jdbc.Driver"
              removeAbandonedOnBorrow="true"
              removeAbandonedOnMaintenance="true"
              removeAbandonedTimeout="60"
              logAbandoned="true"
              minEvictableIdleTimeMillis="300000"
              timeBetweenEvictionRunsMillis="300000"
              url="jdbc:mysql://css244.cs.vt.edu:3306/[Name]BookstoreDB"/>
</Context>
```

[Name] is your name from the "Project Names" Piazza post pinned to the top of your feed. [pid] is your VT PID, and [pin] is the last for digits of your 9-digit VT ID.

Step 6 - Modify index.jsp

Under the webapp directory, replace the code in index.jsp with the following:

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
</head>
<body>
<h1>Database content</h1>
<sql:query var="result" dataSource="jdbc/[Name]Bookstore">
    SELECT * FROM category, book
    WHERE category.category_id = book.category_id
</sql:query>
<table border="1">
    <!-- column headers -->
    <tr>
```

Assignment Project Quiz Exam Essay Help
Wechat: cestbon688
Email: accoder overseas@163.com

```
        <c:forEach var="columnName" items="${result.columnNames}">
            <th><c:out value="${columnName}"/></th>
        </c:forEach>
    </tr>
    <!-- column data -->
    <c:forEach var="row" items="${result.rowsByIndex}">
        <tr>
            <c:forEach var="column" items="${row}">
                <td><c:out value="${column}"/></td>
            </c:forEach>
        </tr>
    </c:forEach>
</table>
</body>
</html>
```

Note that you need to type in your [Name] wherever you see [Name] in red. Note also that the data-source in sql:query does not end in "DB". The datasource name is defined in your context.xml file. **Run your application.**

You should see something like the image below. However, your table will be populated with the categories and books in **your** database.

Database content

category_id	name	book_id	title	author	description	price	rating	is_public	is_featured	category_id
1001	Classics	1001	The Iliad	Homer		699	0	true	false	1001
1001	Classics	1002	The Brothers Karamazov	Fyodor Dostoyevski		799	0	true	true	1001
1001	Classics	1003	Little Dorrit	Charles Dickens		599	0	true	false	1001
1001	Classics	1004	Moby Dick	Herman Melville		699	0	true	false	1001
1001	Classics	1005	Hamlet	William Shakespear		899	0	true	false	1001
1002	Fantasy	1006	Miss Peregrine's Home for Perculiar Children	Ransom Riggs		1299	0	false	false	1002
1002	Fantasy	1007	The Hobbit	J.R.R. Tolkein		1699	0	true	false	1002
1002	Fantasy	1008	Harry Potter and the Sorcerer's Stone	J.K. Rowling		1299	0	false	false	1002
1002	Fantasy	1009	Watership Down	Richard Adams		799	0	true	false	1002
1002	Fantasy	1010	A Game of Thrones: A Song of Ice and Fire	George R.R. Martin		1099	0	false	true	1002
1003	Mystery	1011	The Curious Incident of the Dog in the Night-Time	Mark Haddon		1299	0	false	false	1003
1003	Mystery	1012	The Girl on the Train	Paula Hawkins		1299	0	false	false	1003
1003	Mystery	1013	Murder on the Orient Express	Agatha Christie		799	0	true	true	1003
1003	Mystery	1014	The Woman in White	Wilkie Collins		499	0	true	false	1003
1003	Mystery	1015	The Hounds of the Baskervilles	Arther Conan Doyle		499	0	true	false	1003
1004	Romance	1016	Pride and Prejudice	Jane Austin		699	0	true	true	1004
1004	Romance	1017	The Notebook	Nicholas Sparks		1299	0	false	false	1004
1004	Romance	1018	Gone with the Wind	Margaret Mitchell		799	0	true	false	1004
1004	Romance	1019	Montana Sky	Nora Roberts		999	0	false	false	1004
1004	Romance	1020	The Duke and I	Julie Quinn		1299	0	false	false	1004
1005	Science Fiction	1021	Dune	Frank Herbert		1299	0	false	false	1005
1005	Science Fiction	1022	Foundation	Isaac Asimov		499	0	true	false	1005
1005	Science Fiction	1023	The Handmaid's Tale	Margaret Atwood		899	0	false	false	1005
1005	Science Fiction	1024	The Left Hand of Darkness	Ursala K. LeGuin		699	0	true	true	1005
1005	Science Fiction	1025	Neuromancer	William Gibson		1099	0	false	false	1005

Assignment Project Quiz Exam Essay Help
Wechat: cestbon 688
Email: accounts@163.com overseas

Requirements

database (and therefore your table) should meet the following requirements:

- Have at least four categories
- Have at least four books in all of your categories

- book_id should start from 1001 and increment with each row (it should not skip over numbers)
- Within each category, some is_public values should be true and some should be false
- Categories and books should not be the same as what is in my screenshot (some overlap is okay)

That's it for Part A! Please do not continue to Part B until you have Part A working. If you are experiences any problems, ask on Piazza.

Part B: Creating a REST API using the DAO Pattern

Resources for Part B: dao-business.zip (https://drive.google.com/file/d/1v8c8tUGUMz5nnCZDuJYD5oW1dzALKfip/view?usp=sharing) and rest-api.zip (https://drive.google.com/file/d/1xyX1gzWG8J7jZsqS5pb7WgvNhtUxkj1WX/view?usp=sharing)

In this part of the project, you will create a RESTful API that allows you to access the category and book data that you stored in your database in Part A.

Terminology

Make sure you know what the following terms refer to. Some terms may be discussed here, some my be discussed in the readings, and some you may have to look up online.

- DAO Pattern / DAO class / Model class / JDBC
- Singleton / Immutable
- JAX-RS / Jersey / Jackson
- path parameter / query parameter

Requirements

Typing the following URLs in your address bar should result in you seeing the specified JSON content. Assume the URLs below all have the required prefix. For example, after Step 3, api/categories initially means http://localhost:8080/[Name]BookstoreRest/api/categories (http://localhost:8080/[Name]BookstoreRest/api/categories). After your submission, you should be able to also access https://cs5244.cs.vt.edu:8443/[Name]BookstoreRest/api/categories.

api/categories	all categories
api/categories/1001	first category
api/books/1001	first book
api/categories/1001/books	all books from first category
api/categories/1001/suggested-books	3 random books from first category
api/categories/1001/suggested-books?limit=2	2 random books from first category

These API calls are already implemented for you in the api.zip resource. Notice that they rely on an 'o obtain a category. You will need to implement similar calls that rely on the category name. The will contain a name portion.

- api/categories/name/Mystery

Assignment Project Quiz Exam Essay Help
WeChat: cestbon_688
Email: accoder_overseas@163.com

- api/categories/name/Mystery/books
- api/categories/name/Mystery/suggested-books
- api/categories/name/Mystery/suggested-books?limit=2

Step 1 – Introduction to Jersey

Jersey is a Java framework that serves as the reference implementation of JAX-RS, which is the Java API for RESTful Web Services. Jackson is a Java library the translates Java objects to and from JSON.

To use the methods in ApiResource, Tomcat needs to discover Jersey and Jackson at startup time. The Jersey libraries in build.gradle enable the Tomcat container to scan and find the @ApplicationPath annotation in ApiApplication.java to start the Jersey container at startup time.

```
@ApplicationPath("/api")
public class ApiApplication extends Application {}
```

This tells Tomcat that when it sees a URL with /api/* it should look for the Jersey class ServletContainer, which will indicate what to do.

Step 2 – Import the DAO and API resources

Remove the HelloServlet and any associated folders. Import the files from dao-business.zip (DAO-related classes) and rest-api.zip (API-related classes). Place the api and business folder under the src/main/java directory in your project, so that you have something like this:

Assignment Project Quiz Exam Essay Help
WeChat: cestbon_688
Email: accoder_overseas@163.com

he server does not know how to handle a request, we will send control back to the client ication in the browser - this becomes important when the client knows about the pages in the web application and the server does not - the server only knows API urls. Therefore, create a web.xml file (under webapp/WEB-INF) with the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee https://jakarta.ee/xml/ns/jakartaee/we
b-app_6_0.xsd"
         version="6.0">

    <servlet>
        <servlet-name>Jersey Web Application</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>jersey.config.server.provider.packages</param-name>
            <param-value>api</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Jersey Web Application</servlet-name>
        <url-pattern>/api/*</url-pattern>
    </servlet-mapping>

</web-app>
```

Step 3 – Get Category APIs Working

(setup context.xml and modify JdbcUtils)

To get the project working, you will need to connect to your database.

- In the Java class "business.JdbcUtils", change "DrK" to ***your*** name.

Install the JSON Formatter Chrome extension (https://chromewebstore.google.com/detail/json-formatter/bcjindcccaagfpapjjmafapmmgkkhgoa?hl=en-US&utm_source=ext_sidebar) into your Chrome browser.

Run the project. You should now be able to access the following APIs:

- api/categories
- api/categories/1001

Step 4 – Get a Book API Working

(fix the Book model and application context)

Note: The source code you were given assumes that the book fields added above (description, rating and is_featured) are not part of the book table. Therefore, you will have to add them where necessary in the code.

To get the book APIs working, you need to complete the Book model class. Do that by adding record constructor fields to business.book.Book. **Do NOT add any more methods - our model classes are going to be immutable**. Here is a way to do that using some shortcuts in IntelliJ:

1. Create a record constructor parameter fields for each field in the **book** table
2. Use camelCase for your Java field names (ex: book_id becomes bookId)

Assignment Project Quiz Exam Essay Help
WeChat: cestbon_688
Email: accoder_overseas@163.com

Finally, in business.ApplicationContext, add **BookDao** information (field and getter) similar to how the **CategoryDao** information has been added.

Run the project. In addition to the category APIs, you should now be able to access the following book API:

- api/books/1001

Step 5 – Get other Book APIs Working

(implement additional methods in BookDaoJdbc)

Complete the implementation of **findByCategoryId** in business.book.BookDaoJdbc. **Run the application.** Check that you can access the API:

- api/categories/1001/books

Now complete the implementation of **findRandomByCategoryId** in the same class. **Run the application.** Check that you can access the APIs:

- api/categories/1001/suggested-books
- api/categories/1001/suggested-books?limit=2

In the last URL you are specifying both a category (1001) and a limit (2). The category parameter is part of the path, and therefore it's called a "path parameter". The limit parameter is part of a query, and therefore it is called a "query parameter". Path parameters are used to identify a resource (or resources) and query parameters are used for everything else. For example, a query parameter may be used to indicate how to sort a list of resources, or to restrict a list of resources to only those with a certain property. In our example, we are asking for some random books associated with category ID 1001. The default number of books to return is 3, but ?limit=2 asks to return exactly 2 books.

Step 6 – Get APIs Involving Category Name Working

(add methods to CategoryDaoJdbc and api.ApiResource)

It would be nice if we did not have to know the category ID to look up a category or book resource, so we are going to create new APIs based on category names. Instead of using "categories/{id}" in the API, we will use "categories/name/{category-name}" in the API, where {category-name} will be the category name as it appears in the database. If your category names are capitalized in your database (for example: Mystery), you will have to capitalize them in the path. If a category name is separated by spaces in your database, you will put a space in your path (for example: Science Fiction). That may seem odd to those of you used to substituting %20 for spaces, but most modern browsers are smart enough to make that substitution for us.

plete the implementation of **findByName** in business.category.CategoryDaoJdbc.

After that, complete the following method in api.ApiResource:

Assignment Project Quiz Exam Essay Help
WeChat: cestbon688
Email: accoder overseas@163.com

```
public Category categoryByName(
        @PathParam("category-name") String categoryName,
        @Context HttpServletRequest httpRequest)
```

Once you do, you should be able to access (for example):

- api/categories/name/Science Fiction

Finally, **complete the implementation** of the API methods:

```
public List<Book> booksByCategoryName(...) { ... }
public List<Book> suggestedBooksByCategoryName(...) { ... }
```

Now you should be able to access (for example):

- api/categories/name/Science Fiction/books
- api/categories/name/Science Fiction/suggested-books
- api/categories/name/Science Fiction/suggested-books?limit=2

Upload your project WAR file to the **CS5244 server (http://cs5244.cs.vt.edu:8080/ArchiveUpload/)** and post your URL to Canvas.

Some REST API requirements for our Testing

We will test your REST API using an automated system. Here are some requirements needed to pass our tests.

- a search for all categories returns at least 4 categories
- a search for all categories with IDs 1001-1004 returns a valid category
- a search for all categories with names corresponding to IDs 1001-1004 returns a valid category
- a search for all books in categories with IDs 1001-1004 returns at least 4 books
- a search for all books with IDs 1001-1016 returns a valid book
- a search for all books in categories with names corresponding to IDs 1001-1004 return at least 4 books
- a search for suggested books in categories with IDs 1001-1004 returns exactly 3 books
- a search for suggested books in categories with names corresponding to IDs 1001-1004 returns exactly 3 books
- a search for 1-4 suggested books in categories with IDs 1001-1004 returns the exact number of books requested
- a search for 1-4 suggested books in categories with names corresponding to IDs 1001-1004 returns the exact number of books requested
- all categories returned must contain the following properties (in camel-case, exactly as given here)
  - categoryId, name

  All books returned must contain the following properties (in camel-case, exactly as given here)
  - bookId, title, author, description, price, rating, isPublic, isFeatured, categoryId
- For example the following properties are NOT correct:
  - categoryID, category_id, category_ID, categoryName, public

Assignment Project Quiz Exam Essay Help
WeChat: cestbon688
Email: accoderoverseas@163.com

http://cs5244.cs.vt.edu:8080/JinhengBookstoreRest/

New Attempt

Assignment Project Quiz Exam Essay Help
WeChat: cestbon_688
Email: accoder_overseas@163.com