**Project 2**

**Spring JDBC, Spring Data JPA, Spring MyBatis, and Beginning Spring MVC**

**Project Preparation**

For this project we are going to be using the same dataset as Project 1. Start by creating three Eclipse **Maven** projects similar to the way we did in Project 1 (**see below**). Name the projects as follows:

1. P2\_2013Q4\_House\_Disburse\_JdbcTemplate
2. P2\_2013Q4\_House\_Disburse\_MyBatis
3. P2\_2013Q4\_House\_Disburse\_JPA

These are going to be **Spring Boot** projects. Create the following package structure in each of the projects:

* **config** package
* **disburse** package
* **disburse.vo** package
* **disburse.controller** package
* **disburse.dao** package

For the JdbcTempate project, do not create any additional packages.

For the **MyBatis project**, create one additional package:

* **disburse.mapper**

For the **JPA project**, create one additional package:

* **disburse.repository**

Add the included file: **2013Q4\_HOUSE\_DISBURSE.sql** to the **src/main/resources** folder **for all projects.**

Use the same **dependencies** from the Maven POM that we use in our in-class **CRM** Project for all projects.

In this project you will be completing a set of challenges. Please read the instructions carefully.

**Project Challenges**

**Section I – Configuration**

1. **Code Challenge**:

**For all projects:** Create a Class in the **disburse.vo** package in each of the projects called **HouseDisburseDetail**. Create member variables (variables at the class level) using the fields from the table “**T\_2013Q4\_HOUSE\_DISBURSE**” in the **2013Q4\_HOUSE\_DISBURSE.sql** file.Do **NOT** use the same naming conventions as in Project 1. Also, the **sql file has changed** from Project 1. So, when creating your member variables, observe the new fields in the file before creating member variables. In this project we are going to use standard[**camel case**](https://techterms.com/definition/camelcase)naming conventions where the first letter of each word is lower case, and no underscores**.** Make sure to set the data types correctly. (varchar = String, INTEGER = int, DECIMAL = double). **For example, I would like you to use for BIOGUIDE\_ID** **field**:

**private String bioGuideID;**

**For all projects**: In the **HouseDisburseDetail** class that was just created, generate “getters and setters” in Eclipse. Also, generate a “toString()” method.

1. **Code Challenge:**

**For all projects:** Create a class in the **disburse** package called “**HouseDisburseDetailApp**”. Make sure to include the stubs for the **public static void main()** method as we did in Project 1. However, **unlike** Project 1, we need to code this class as a **Spring Boot** application. (Like we did for our demo project in class. We’ll be creating the config class shortly.)

1. **Code Challenge:** *(Note: On this step you will not be performing the same action for all projects. Please pay close to attention.)*

**For the MyBatis Project:**

Create a mapper **interface** called **HouseDisburseMapper** in the **disbuse.mapper** package. Make sure to annotate this **interface** using the MyBatis mapper annotation.

**For the JPA Project:**

Create a Repository **interface** called **HouseDisburseRepository** in the **disburse.repository** package. Ensure this **extends** the correct JpaRepository and type. (Hint: Remember the first value in the extends syntax needs to be POJO you want to map to. The second is the data type of the Id! (And the data type of the ID must be an object class and not a primitive type. So do not use “**int**” for example, use **Integer**)

1. **Code Challenge:**

**For all projects:** Create a class in the **disburse.dao** package called **HouseDisburseDAO.**

***Pay close to attention to the following:***

**For all projects:** This class needs to be made into a Spring Bean. Use Annotation Config or Java Config in the config class we will make later.

**For the JdbcTemplate project:** Inject a **JdbcTemplate** bean into the **HouseDisburseDAO** class using **Constructor** injection ***only***. If you are confused about this step, go here: <https://reflectoring.io/constructor-injection/>. I will be taking points off if you use some other method of injection.

**For the MyBatis project:** Inject **HouseDisburseMapper** into the **HouseDisburseDAO** class using **Field** Injection ***only***. If you are confused about this step, go here: <https://reflectoring.io/constructor-injection/>. I will be taking points off if you use some other method of injection.

**For the JPA project:** Inject **HouseDisburseRepository** into the **HouseDisburseDAO** class using **Setter** Injection ***only.*** If you are confused about this step, go here: <https://reflectoring.io/constructor-injection/>. I will be taking points off if you use some other method of injection.

1. **Code Challenge:**

**For all projects:** Create a class in the **disburse.controller** package called **HouseDisburseController**. This class needs to be made into a Spring Bean. Use Annotation Config or JavaConfig. You must inject the **HouseDisburseDAO** bean into this class. You may use **any** injection method.

1. **Code Challenge:**

**For all projects**:In the **config** package create a class called **HouseDisburseConfig**. Annotate this class as a **Spring Boot** application.

**For all projects:**

1. Create a **DataSource** Spring Bean using our normal EmbeddedDatabaseBuilder (HyperSQL Database) and building the attached .sql file.
2. If you are using AnnotationConfig in any of your previously created classes, you must use the correct annotation to scan these classes in this class.
3. Return to the **HouseDisburseDetailApp** and add code to run the app as SpringApplication under the main method calling the **HouseDisburseConfig** class.

**For the MyBatis project:**

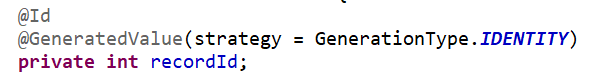
1. Use the appropriate annotation to scan for the Mybatis mapper interface.
2. Create a SqlSessionFactory Bean. (Examine our in class SpringMVCDemo project to do this. Make sure to change the .setTypeAliasesPackage to scan **our** project’s value object or POJO (**vo**) package)

**For the JPA project:**

1. Use the appropriate annotation to scan for the Jpa Repositories.
2. Create a **LocalContainerEntityManagerFactoryBean.** Make to set the .setPackagesToScan to our project’s value object or POJO (**vo**) package.
3. **Code Challenge:**

**For the JPA Project**:

1. Return to the **HouseDisburseDetail.** Use the correct JPA annotation to the label the Class as a JPA Entity and Table the class is mapped to.
2. Label the recordId field with the @Id annotation **AND** the @GeneratedValue annotation exactly as below:



1. Label all the member variables with the correct @Column annotation to map the Column names in the database to the fields in the POJO.
2. **Code Challenge:**

**For all Projects:**

1. Create a folder in the **src/main/resources** pathcalled “**templates**”. This is a ThymeLeaf folder. The folder MUST be called “templates”.
2. Create a HTML file called “**disburse.html**”
3. Create a html structure like the following. Pay very close attention to the variable name “hdList” and notice the properties such as **hd.bioGuideID** correspond to member variable names in the **HouseDisburseDetail** class.



**Section II** - **Spring MVC and Spring Data Utilization**

Develop solutions to display results of the following requests in the browser. **Unless instructed different**, display the results in the **ThymeLeaf page.**

1. **Code Challenge:**

**For all Projects.**

**When I run the project(s) and access:** <http://localhost:8080/all>

**I want to see in the browser:**

1. All records and fields in the entire datatable.
2. Convert all letters from the **OFFICE** column to display in **LOWER CASE** in the browser.
3. **Hint:** Convert all letters to lower case in the DAO class before it comes back to the controller and displays in the browser. (Remember: This is what the DAO class is for. To serve as a middle layer to massage data that comes back from mappers. Do not do this logic in the controller class)
4. Display the results in the **ThymeLeaf** page.
5. **Hints:** If you’re seeing “null” or empty values in the table, then make sure the properties are being mapped correctly. For JdbcTemplate if you’re using **BeanPropertyRowMapper** the SQL statement may need to read something like “SELECT BIOGUIDE\_ID as bioGuideID, OFFICE as office….”. Or use a **RowMapper**. For MyBatis you may have to use the @Results annotation. For JPA make sure to use the @Column annotations.

**Perform this in ALL three projects.**

1. **Code Challenge:**

**For all Projects.**

**When I run the project(s) and access from the browser:** [**http://localhost:8080/bioGuideID/{BIOGUIDE\_ID}**](http://localhost:8080/bioGuideID/%7bBIOGUIDE_ID%7d) **where {BIOGUIDE\_ID} is a BIOGUIDE\_ID value of my choosing I will try to run. For example, I *could* attempt to run** [**http://localhost:8080/bioGuideID/A000055**](http://localhost:8080/bioGuideID/A000055) **)**

**Hint: IMPORTANT. Notice the URL is NOT at the root so make sure your mapping reflects “bioGuideID/{variable\_name}”.**

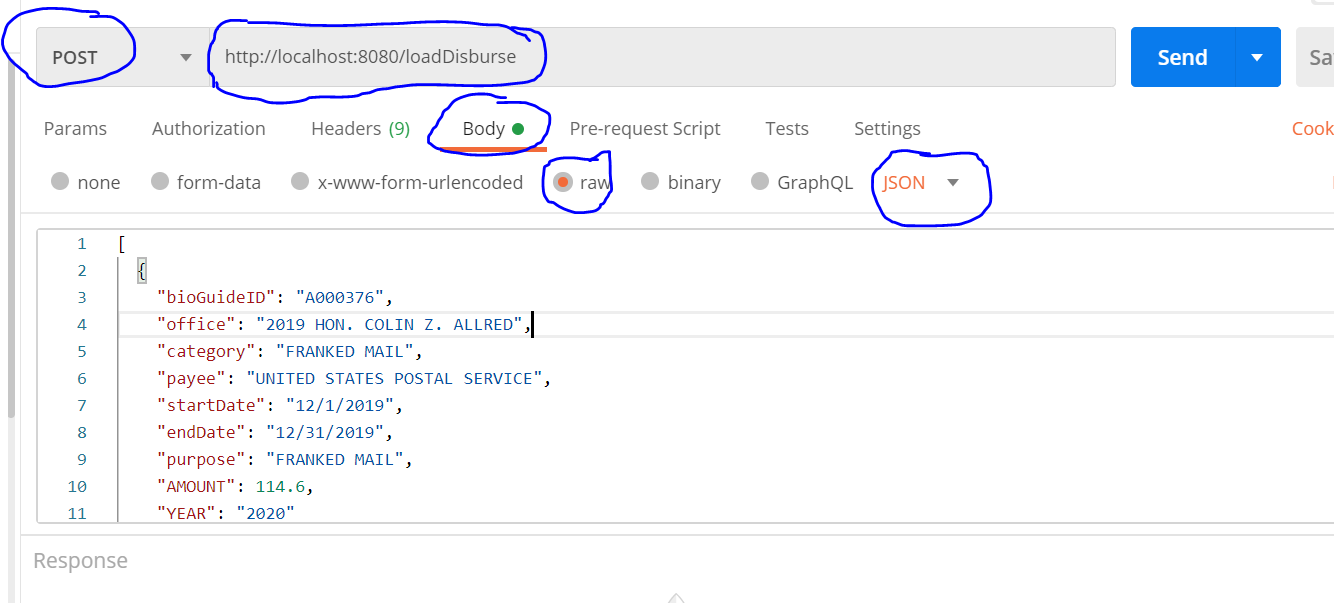
**I want to see in the browser:** All records and fields with ONLY the BIOGUIDE\_ID I choose. Display in the Thymeleaf page.

**This must be done for ALL projects.** In the JPA Project you must use the **findByXXXX** method.

**Code Challenge:**

**For all projects.**

For this challenge you will have to use Postman and the accompanying file **HouseDisburseJSON.txt** to test the results**.** Open Postman and send a **POST** request to <http://localhost:8080/loadDisburse> and paste the in the **Body** of the request the contents of the **HouseDisburseJSON.txt** file. See screenshot:



**Hence:**

**When I run the project(s), and then send a POST request to** <http://localhost:8080/loadDisburse> **with the Request Body from the contents of the HouseDIsurseJSON.txt file:**

1. I want the records I sent in the **POST** request to be saved in the Database. Make sure you annotate your code to accept Post Mapping only.
2. I ALSO want to see JUST the records in the database to be sent back as a **Response Body**. **Use a query to return the records for the Response Body. Hint: (Notice the records inserted are from YEAR 2020) Do not display** this in the ThymeLeaf html page**.**

**IMPORTANT:** You **SHOULD NOT** be seeing any null values returned. **Look closely at the JSON in the file. Do the key names match your member variable names?** You may need to use **@JsonProperty.** In the MyBatis project, remember, you may have to additionally use the **@Results** annotation**.**

**This must be done for ALL Projects.**

1. **Code Challenge:**

**For the JdbcTemplate and JPA projects only. (No MyBatis. Although it is possible in a similar way we did the Batch Insert)**

**When I run the project(s) and access the URL at** [**http://localhost:8080/changeCategory**](http://localhost:8080/changeCategory) **--**

1. **Update** all the records in the database such that the **CATEGORY** column values are updated according to the mapping in the table below. Update all records for the **CATEGORY** column from the “old” value to the “new” value. For every instance of the “old” value, replace it with the “new” value. (This does not need to include the results from Code Challenge 3)

Look at your **2013Q4\_House\_DISBURSE.sql file. You’ll notice that these are the values in the third column.** Make sure that your **update is done in a Batch Update execution**. I **don’t** want to see individual Update statements executed in a loop.

1. After the records are updated display all records (with updated results) in the ThymeLeaf page. Use a query.

|  |  |
| --- | --- |
| **Old Value** | **New Value** |
| TRAVEL | T |
| FRANKED MAIL | FM |
| PERSONNEL COMPENSATION | PC |
| RENT | R |
| EQUIPMENT | E |
| OTHER SERVICES | OS |
| SUPPLIES AND MATERIALS | SM |
| PRINTING AND REPRODUCTION | PR |

**For example, this record:**

'A000022','HON. GARY L. ACKERMAN','**TRAVEL**','CITIBANK','12/28/12','12/28/12','COMMERCIAL TRANSPORTATION','-78.7','2012'

**Will be updated to this:**

'A000022','HON. GARY L. ACKERMAN','**T**','CITIBANK','12/28/12','12/28/12','COMMERCIAL TRANSPORTATION','-78.7','2012'

1. **Code Challenge:**

**For all projects.**

**When I run the project(s) and access from the browser:** [**http://localhost:8080/highestAmount**](http://localhost:8080/highestAmount)

**I want to see in the browser:** the record or records that has/have the highest AMOUNT spent. This may be displayed as a Response Body (JSON) or in the Thymeleaf page.

**Hint:** Your query will need to be: SELECT \* from T\_2013Q4\_HOUSE\_DISBURSE where AMOUNT = (SELECT MAX(AMOUNT) from T\_2013Q4\_HOUSE\_DISBURSE)

**This must be done for ALL projects.**

1. **Code Challenge:**

**For the JdbcTemplate Project ONLY.**

**When I run the project(s) and access from the browser:** [**http://localhost:8080/highestAmountByBioGuideID**](http://localhost:8080/highestAmountByBioGuideID)

**I want to see in the browser:** A list key/value pairs of BIOGUIDE\_ID and AMOUNT. This may be displayed as a Response Body. (JSON)

**Hint:**

Your query will need to be: **SELECT BIOGUIDE\_ID, MAX(AMOUNT) from T\_2013Q4\_HOUSE\_DISBURSE GROUP BY BIOGUIDE\_ID**

Store this value as a **List<Map<String, Object>>** type and return it in a **Response Body**.

**This must be done for JdbcTemplate only.**