Spring MVC Tutorial – Contact List Application

Step 16: Implementing Stats REST Controller Endpoints

SOFTWARE GUILD

The Learning House

427 S 4th Street #300

Louisville KY 40202

Step 16: Implementing Stats REST Controller Endpoints

Overview

In this step, we will create a new REST endpoint in the Stats controller. This endpoint will use the Google Java Visualization library to return a Google DataTable JSON object that can be displayed on the client using the Google Charts JavaScript API. This feature requires the following changes:

- Add dependency to POM
- Create new DTO for stats information
- Add a method to the DAO interface
- Implement new method in concrete DAO implementations
- Add REST endpoint to StatsController

Modifying the POM

To use the Google Visualization library, we must update our POM to include this dependency:

```
<dependency>
     <groupId>com.google.visualization</groupId>
     <artifactId>visualization-datasource</artifactId>
     <version>1.1.1</version>
</dependency>
```

Create New DTO

We need a new DTO to hold Contact statistics retrieved from the database. Each DTO holds a company name and the number of contacts for that company. Create a new Java class called **CompanyContactCount** in the **model** package and add the following code:

```
package com.swcguild.contactlistmvc.model;


public class CompanyContactCount {

    private String company;

    private int numContacts;


    public String getCompany() {

        return company;

    }


    public void setCompany(String company) {

        this.company = company;

    }


    public int getNumContacts() {

        return numContacts;

    }


    public void setNumContacts(int numContacts) {

        this.numContacts = numContacts;

    }

}
```

Modifying the DAO Interface

We need another method in our DAO that will allow us to retrieve company count information from the database. Add the following method to your DAO interface (you will have to import CompanyContactCount). This method returns a list of CompanyContactCount DTO objects; the list will be converted into a Google DataTable object by the controller.

```
public List<CompanyContactCount> getCompanyContactCounts();
```

Modifying the DAO Implementations

We must also modify the DAO implementations to match the DAO interface. Add the following code to ContactListDaoInMemImpl.java (because we have moved on to the database implementation of the DAO interface, we leave the in-memory DAO implementation of this method to the reader):

```
@Override

public List<CompanyContactCount> getCompanyContactCounts() {

        throw new UnsupportedOperationException("Not supported yet.");

}
```

Modifying the database implementation of the DAO requires more extensive changes. Following the pattern established for creating, reading, updating, and deleting Contact objects, we have to do the following:

1. Implement getCompanyContactCounts()
2. Add a prepared statement to retrieve company count data from the database
3. Create a RowMapper to map data from the database to CompanyContactCount DTOs

Add the following code to ContactListDaoDbImpl.java:

Method:

```
@Override
public List<CompanyContactCount> getCompanyContactCounts() {
        return jdbcTemplate.query(SQL_SELECT_COMPANY_CONTACT_COUNTS,
                                  new CompanyContactCountMapper());
}
```

Prepared Statement:

```
private static final String SQL_SELECT_COMPANY_CONTACT_COUNTS
       = "SELECT company, count(*) as num_contacts FROM contacts group by company;";
```

Row Mapper:

```
private static final class CompanyContactCountMapper
    implements ParameterizedRowMapper<CompanyContactCount> {

    @Override
    public CompanyContactCount mapRow(ResultSet rs, int i) throws SQLException {
        CompanyContactCount count = new CompanyContactCount();
        count.setCompany(rs.getString("company"));
        count.setNumContacts(rs.getInt("num_contacts"));
        return count;
    }
}
```

Designing Our URL:

We will design the endpoint for Stats following the same approach used for the endpoints created in previous steps:

Searching for Contacts

- **Verb:** GET
- **URL:** stats/chart
- **RequestBody:** None
- **ResponseBody:** Google DataTable JSON object containing Contact chart data

Controller Implementation

Next, we will create code in the controller that maps our URL, retrieves company contact data from the database, and converts the data to a Google DataTable object.

Add the following method to your StatsController (see code comments for details):

```
@RequestMapping(value = "/stats/chart", method = RequestMethod.GET)
// the @ResponseBody annotation tells Spring MVC that the value returned from
// this method is NOT the name of a View component.  The value returned should
// be the body of the response to the caller
@ResponseBody public String getDataTable() {
     try {
         // Get the contact counts for each company
         List<CompanyContactCount> counts = dao.getCompanyContactCounts();

         // Set up the table columns and descriptions
         DataTable t = new DataTable();
         t.addColumn(new ColumnDescription("Company_Name",
                                           ValueType.TEXT,
                                           "Company"));
         t.addColumn(new ColumnDescription("Number_Contacts",
                                           ValueType.NUMBER,
                                           "# Contacts"));
         // Convert each CompanyContactCount object into a table row
         for(CompanyContactCount currentCount : counts) {
             TableRow tr = new TableRow();
             tr.addCell(currentCount.getCompany());
             tr.addCell(currentCount.getNumContacts());
             t.addRow(tr);
         }
         // Use the JsonRenderer to convert the DataTable to a JSON string
         // the default Jackson converter doesn't convert the DataTable object
         // to JSON properly so we have to do it here.
         return JsonRenderer.renderDataTable(t, true, false, false).toString();
     } catch (Exception e) {
         // just return an error string if we encounter an issue
         return "Invalid Data";
     }
 }
```

Wrap-up

In this step, we created a REST endpoint to supply data to our stats chart. In the next step, we'll wire the chart to live data.