Spring MVC Tutorial – Contact List Application

Step 03: Creating the DAO Interface and Model Components

SOFTWARE GUILD

The Learning House

427 S 4th Street #300

Louisville KY 40202

Step 03: Creating the DAO Interface and Model Components

Overview

In this step, we will start to build some server-side functionality by creating the DAO interface and model components for our project. We have already done some design work on these components in Step 00. This upfront design work will help ensure that the model objects we create in Java on the server side will match up properly to the JavaScript objects we created in the previous steps. The design that we created in Step 00 also dictates a minimum level of functionality for our data layer.

Model Component

As specified in our design, the model for this application consists of one class that holds the first name, last name, company, phone, and email of a contact. Create the following class in the com.swcguild.contactlistmvc.model package:

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

import java.util.Objects;

public class Contact {

    private int contactId;
    private String firstName;
    private String lastName;
    private String company;
    private String phone;
    private String email;

    public int getContactId() {
        return contactId;
    }
    public void setContactId(int contactId) {
        this.contactId = contactId;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
```

```
}
public String getCompany() {
    return company;
}

public void setCompany(String company) {
    this.company = company;
}
public String getPhone() {
    return phone;
}
public void setPhone(String phone) {
    this.phone = phone;
}
public String getEmail() {
    return email;
}
public void setEmail(String email) {
    this.email = email;
}

@Override
public int hashCode() {
    int hash = 5;
    hash = 37 * hash + this.contactId;
    hash = 37 * hash + Objects.hashCode(this.firstName);
    hash = 37 * hash + Objects.hashCode(this.lastName);
    hash = 37 * hash + Objects.hashCode(this.company);
    hash = 37 * hash + Objects.hashCode(this.phone);
    hash = 37 * hash + Objects.hashCode(this.email);
    return hash;
}
```

```
    @Override
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Contact other = (Contact) obj;
        if (this.contactId != other.contactId) {
            return false;
        }
        if (!Objects.equals(this.firstName, other.firstName)) {
            return false;
        }
        if (!Objects.equals(this.lastName, other.lastName)) {
            return false;
        }
        if (!Objects.equals(this.company, other.company)) {
            return false;
        }
        if (!Objects.equals(this.phone, other.phone)) {
            return false;
        }
        if (!Objects.equals(this.email, other.email)) {
            return false;
        }
        return true;
    }
}
```

DAO Interface

The first step in creating a data layer for our application is to define an interface for our DAO. In order to ensure that the layers of our application are loosely coupled, we always program to interfaces, never to concrete implementations. As we will demonstrate in later steps of this tutorial, programming to the DAO interface will allow us to easily switch between concrete DAO implementations without having to change any of the Java code that uses the DAO.

The application that we are building has basic create, retrieve, update, and delete (CRUD) functionality for contact information. Our interface will allow retrieval of contact information by ID or by some combination of first name, last name, company, phone, and email search terms.

Our design document specifies the following search behavior:

1. If no search criteria are entered, all contacts will be returned.
2. The user can enter one or more fields for search. The application will perform an AND search with the given terms. For example, if the user enters “Smith” for last name and “Apple” for company, the system will return all contacts with the last name of Smith who work at Apple.

Our search method will take a Map parameter that contains the criteria that will be used to perform the search. We want tight control over what types of search criteria the user can specify — the only things that make sense are First Name, Last Name, Company, Phone, and Email. For example, it is meaningless to search for Contacts where Hometown = Chicago because we don’t keep track of Hometown. To facilitate this control, we will create an enumerated type and then use this enumerated type for the keys in our criteria map.

Create a new Java enum called **SearchTerm** in the **dao** package and enter the following code:

```
package com.swcguild.contactlistmvc.dao;

public enum SearchTerm {
    FIRST_NAME, LAST_NAME, COMPANY, PHONE, EMAIL
}
```

Now, create the following interface in the com.swcguild.contactlistmvc.dao package:

```
package com.swcguild.contactlistmvc.dao;

import com.swcguild.contactlistmvc.model.Contact;
import java.util.List;
import java.util.Map;

public interface ContactListDao {
// add the given Contact to the data store

    public Contact addContact(Contact contact);

    // remove the Contact with the given id from the data store
    public void removeContact(int contactId);

    // update the given Contact in the data store
    public void updateContact(Contact contact);

    // retrieve all Contacts from the data store
    public List<Contact> getAllContacts();

    // retrieve the Contact with the given id from the data store
    public Contact getContactById(int contactId);

    // search for Contacts by the given search criteria values
    public List<Contact> searchContacts(Map<SearchTerm, String> criteria);
}
```

Note that the keys for our criteria Map are of type SearchTerm. This guarantees that callers will only be able to specify valid search criteria.

Wrap-up

That completes this portion of the tutorial. In the next step, we will create the in-memory version of the DAO.