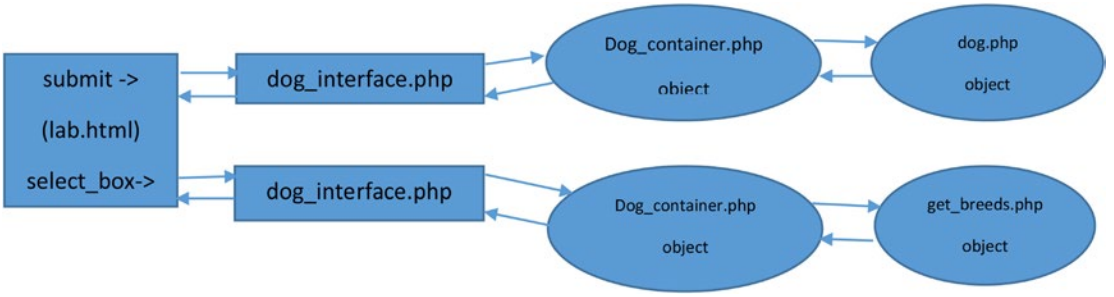


***Figure 4-5.*** *Data flow for the dog application*

The `dog_interface.php` program will provide the interface for all parts of the dog application. In addition it provides security and filtering as you have already designed in the previous examples. Along with these other activities, `dog_interface` creates and uses the `dog_container` object to contain, create, and pass any other objects needed (without knowing the name of the objects). The `dog_container` object uses an XML file (not shown) to discover the location and name of files containing the classes it will create (providing dependency injection).

The application will always use the `dog_interface` program to access the other classes. The `dog_interface` program will determine what classes are needed to accomplish a particular task. Whenever a class is needed, the `dog_interface` will use the `dog_container` to determine the name and location of the class (via the XML file), and to create an instance of the class (object). By using an XML file to list the class file names and locations, changes can be made without causing any code changes to programs in the application.

When the breed select box is requested from the `lab.html` page, the `dog_interface` program is called. It will create a `dog_container` object. The `dog_container` object will discover the location and file name of the `get_breeds` class file and the breeds XML file. Once it is discovered, the `dog_container` object will create a `get_breeds` object. The `get_breeds` object will then build the code for the select box; eventually returning the code to the form in `lab.html` for display to the user. All objects (`dog_container` and `get_breeds`) are then destroyed (removed from memory).

When the Submit button is clicked on the `lab.html` form (assuming all validation is passed), it will call the `dog_interface` program. This program will create the `dog_container` object. The `dog_container` object will then discover the location and file name of the `dog.php` class file. Once discovered, the `dog_container` object will create the `dog` object. The `dog` object will behave exactly as it has in previous examples (validating properties and displaying properties). Once the `dog` object has completed, the objects (`dog_container` and `dog`) are destroyed (removed from memory). This design allows complete independence of the objects providing both two-tier design (interface and business rules) and dependency injection.

***Example 4-10.*** The dog_applications.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<dog_applications>
<application>
<type ID="dog">
<location>dog3.php</location>
</type>
</application>
<application>
<type ID="selectbox">
<location>get_breeds2.php</location>
</type>
</application>
```