Tapestry5MultipleSelectOnObjects

MultipleSelect Component

Tapestry 5 doesn´t contain a multiple select component yet. So here is an implementation for a generic multiple select based on Daniel Jue's GenericSelectionModel and GenericValueEncoder classes and some modified base components. I leaved the unmodified parts in the code for better copy and paste.

> Note: This renders as a Select element, so users have to control-click or shift-click to select multiple items. A better choice for most purposes is the Palette component or (for Tapestry 5.3) the Checklist component.

See also :

- Tapestry5 How to use the SELECT component

SelectMultiple.java

First we need a modified implementation of the core select class which i named SelectMultiple. You can store it anywhere in your package system.

```
import java.util.List;
import java.util.Locale;

import org.apache.tapestry.Binding;
import org.apache.tapestry.ComponentResources;
import org.apache.tapestry.FieldValidator;
import org.apache.tapestry.MarkupWriter;
import org.apache.tapestry.OptionModel;
import org.apache.tapestry.SelectModel;
import org.apache.tapestry.SelectModelVisitor;
import org.apache.tapestry.ValidationException;
import org.apache.tapestry.ValidationTracker;
import org.apache.tapestry.ValueEncoder;
import org.apache.tapestry.annotations.BeforeRenderTemplate;
import org.apache.tapestry.annotations.Environmental;
import org.apache.tapestry.annotations.Inject;
import org.apache.tapestry.annotations.Parameter;
import org.apache.tapestry.corelib.base.AbstractField;
import org.apache.tapestry.services.FieldValidatorDefaultSource;
import org.apache.tapestry.services.FormSupport;
import org.apache.tapestry.services.Request;
import org.apache.tapestry.services.ValueEncoderFactory;
import org.apache.tapestry.services.ValueEncoderSource;
import org.apache.tapestry.util.EnumSelectModel;

import pathToYourPackageSystem.models.MultiValueEncoder;
import pathToYourPackageSystem.models.SelectMultipleModelRenderer;

public final class SelectMultiple extends AbstractField
{
    private class Renderer extends SelectMultipleModelRenderer
    {

        public Renderer(MarkupWriter writer)
        {
            super(writer, _encoder);
        }


        @Override
        protected boolean isOptionSelected(OptionModel optionModel)
        {
                Object value = optionModel.getValue();
                return (_values == null) ? false : _values.contains(value);
        }

    }

    @Parameter
    private MultiValueEncoder _encoder;
```

```
@Inject
private FieldValidatorDefaultSource _fieldValidatorDefaultSource;

@Inject
private Locale _locale;

@Parameter(required = true)
private SelectModel _model;

@Inject
private Request _request;

@Inject
private ComponentResources _resources;

@Environmental
private ValidationTracker _tracker;

/** Performs input validation on the value supplied by the user in the form submission. */
@Parameter(defaultPrefix = "validate")
@SuppressWarnings("unchecked")
private FieldValidator<Object> _validate = NOOP_VALIDATOR;

/** The list of value to read or update. */
@Parameter(required = true, principal = true)
private List<Object> _values;


@Inject
private ValueEncoderSource _valueEncoderSource;

@Override
@SuppressWarnings("unchecked")
protected void processSubmission(FormSupport formSupport, String elementName)
{
        String[] primaryKeys = _request.getParameters(elementName);
        List<Object> selectedValues = _encoder.toValue(primaryKeys);

        try
    {
        for (Object selectedValue : selectedValues) {
                        _validate.validate(selectedValue);
        }
            _values = selectedValues;

    }
    catch (ValidationException ex)
    {
        _tracker.recordError(this, ex.getMessage());
        return;
    }
}

void afterRender(MarkupWriter writer)
{
    writer.end();
}

void beginRender(MarkupWriter writer)
{
    writer.element("select", "name", getElementName() , "id", getClientId(), "multiple", "multiple");
}

@SuppressWarnings("unchecked")
ValueEncoder defaultEncoder()
{
    return _valueEncoderSource.createEncoder("value", _resources);
}

@SuppressWarnings("unchecked")
```

```
    SelectModel defaultModel()
    {
        Class valueType = _resources.getBoundType("value");

        if (valueType == null) return null;

        if (Enum.class.isAssignableFrom(valueType))
            return new EnumSelectModel(valueType, _resources.getContainerMessages());

        return null;
    }

    FieldValidator defaultValidate()
    {
        Class type = _resources.getBoundType("value");

        if (type == null) return null;

        return _fieldValidatorDefaultSource.createDefaultValidator(
                this,
                _resources.getId(),
                _resources.getContainerMessages(),
                _locale,
                type,
                _resources.getAnnotationProvider("value"));
    }

    Binding defaultValue()
    {
        return createDefaultParameterBinding("value");
    }

    @BeforeRenderTemplate
    void options(MarkupWriter writer)
    {
        SelectModelVisitor renderer = new Renderer(writer);

        _model.visit(renderer);
    }

    // For testing.

    void setModel(SelectModel model)
    {
        _model = model;
    }

    void setValues(List<Object> values)
    {
        _values = values;
    }

    void setValueEncoder(MultiValueEncoder encoder)
    {
        _encoder = encoder;
    }
}
```

As you can see, we need a multiple value encoder interface and a multiple select model renderer, so here they are.

MultipleValueEncoder.java

```
import java.util.List;

import org.apache.tapestry.SelectModel;
import org.apache.tapestry.corelib.components.Select;

public interface MultiValueEncoder<V>
{
    List<String> toClient(V value);

    List<V> toValue(String[] clientValue);
}
```

SelectMultipleModelRenderer.java

```
import java.util.Map;
import java.util.List;

import org.apache.tapestry.MarkupWriter;
import org.apache.tapestry.OptionGroupModel;
import org.apache.tapestry.OptionModel;
import org.apache.tapestry.SelectModelVisitor;

public class SelectMultipleModelRenderer implements SelectModelVisitor
{
    private final MarkupWriter _writer;

    private final MultiValueEncoder _encoder;

    public SelectMultipleModelRenderer(final MarkupWriter writer, MultiValueEncoder encoder)
    {
        _writer = writer;
        _encoder = encoder;
    }

    public void beginOptionGroup(OptionGroupModel groupModel)
    {
        _writer.element("optgroup", "label", groupModel.getLabel());

        writeDisabled(groupModel.isDisabled());
        writeAttributes(groupModel.getAttributes());
    }

    public void endOptionGroup(OptionGroupModel groupModel)
    {
        _writer.end(); // select
    }

    @SuppressWarnings("unchecked")
    public void option(OptionModel optionModel)
    {
        Object optionValue = optionModel.getValue();

        List<String> clientValues = _encoder.toClient(optionValue);

        for (String clientValue : clientValues) {
                _writer.element("option", "value", clientValue);

                if (isOptionSelected(optionModel)) _writer.attributes("selected", "selected");

                writeDisabled(optionModel.isDisabled());
                writeAttributes(optionModel.getAttributes());

                _writer.write(optionModel.getLabel());
        }

        _writer.end();
    }
```

```
    private void writeDisabled(boolean disabled)
    {
        if (disabled) _writer.attributes("disabled", "disabled");
    }

    private void writeAttributes(Map<String, String> attributes)
    {
        if (attributes == null) return;

        for (Map.Entry<String, String> e : attributes.entrySet())
            _writer.attributes(e.getKey(), e.getValue());
    }

    protected boolean isOptionSelected(OptionModel optionModel)
    {
        return false;
    }

}
```

Now we need the implementation of the modified generic value encoder. In case of no selected values it will return an empty list.

GenericMultipleValueEncoder.java

```
import java.lang.reflect.InvocationTargetException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import org.apache.commons.beanutils.BeanUtils;
public class GenericMultiValueEncoder<T> implements MultiValueEncoder<T> {

	private List<T> list;

	private String labelField;

	public GenericMultiValueEncoder(List<T> list, String labelField) {
		this.list = list;
		this.labelField = labelField;
	}

	public List<String> toClient(T obj) {
		try {
			return Arrays.asList(BeanUtils.getArrayProperty(obj, labelField));
		} catch (IllegalAccessException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (InvocationTargetException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (NoSuchMethodException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		return null;
	}

	public List<T> toValue(String[] strings) {
		try {
			List<String> test = Arrays.asList(strings);
			List<T> valueList = new ArrayList<T>();
			for (T obj : list) {
				if (test.contains(BeanUtils.getProperty(obj, labelField))) {
					valueList.add(obj);
				}
			}
			return valueList;
		} catch (NullPointerException e) {
			List<T> exceptionReturn = new ArrayList<T>();
			return exceptionReturn;
		} catch (IllegalAccessException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (InvocationTargetException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (NoSuchMethodException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		return null;
	}
}
```

Usage

Now you can use the multiple select in your pages. First you need a List parameter in your page class. You will need the GenericSelectionModel and the GenericMultiValueEncoder also.

```
...
@Persist // Could be @Parameter if needed
private List<DemoObject> dOs;

public List<DemoObject> getDOs() {
        return dOs;
}

public void setDOs(List<DemoObject> new_dOs) {
        dOs = new_dOs;
}
...
public GenericSelectionModel<dO> getDOSelectModel() {
        return new GenericSelectionModel<dO>(your_model.method(),
                        "your_label");
}

public GenericMultiValueEncoder<dO> getDOValueEncoder() {
        return new GenericMultiValueEncoder<dO>(your_model.method(), "your_id");
}
...
```

The page template has to contain following code inside the form.

```
...
<t:selectmultiple values="dOs"
                  model="DOSelectModel"
                  encoder="DOValueEncoder"
/>
...
```

Note that this implementation can´t be used with the automatic generated encoder for enums. In this case you have to implement the enum encoder manually.