LIFERAY

BUILDING APPLICATION UIS WITH SOY TEMPLATES

REQUIREMENTS

- S.P.A.C.E. would like to add a new application for students and instructors to manage an activity list.
- The development team has been tasked with creating this new application.
- The back-end developers have created a new set of APIs using Liferay Service Builder and have exposed JSON endpoints to these services.
- They have also decided to use a SoyPortlet to render this new application.
- As front-end-developers, we need to build a nice UI for this new application.

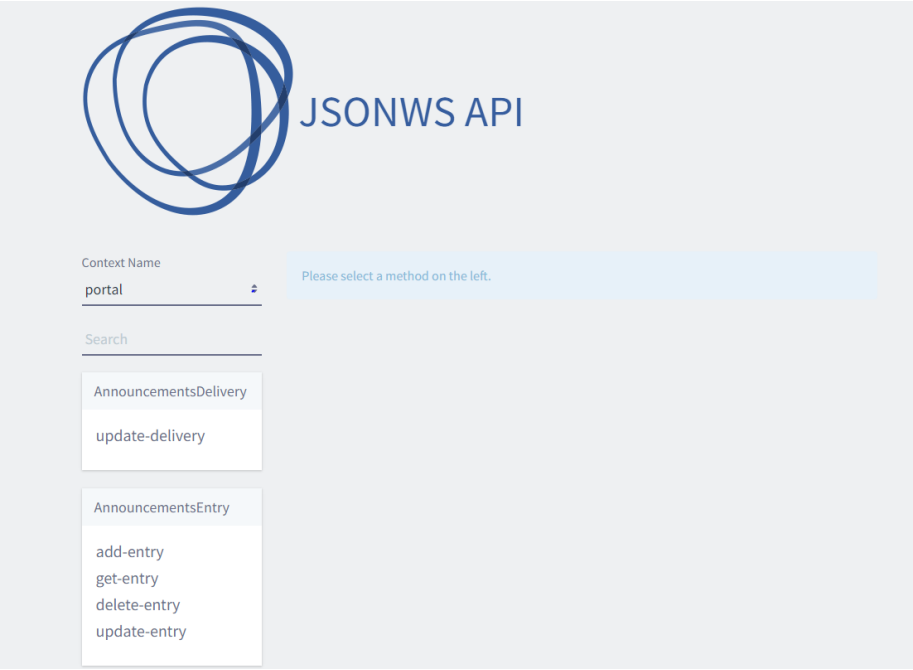
EXERCISE: API

- First, let's take a look at this new API and see what is available to us.
- For this exercise, you'll need all the files available in your `appendix-soy-templates` directory.

1. **Go to** *exercises/front-end-developer-exercises/appendix-soy-templates*.
2. **Copy** the *com.liferay.todo.api.jar* and the *com.liferay.todo.service.jar*.
3. **Go to** your *[LIFERAY_HOME]* directory here: *liferay/bundles/liferay-dxp-digital-enterprise-[version]*.
4. **Paste** the files in the */deploy* folder.
5. **Go to** *http://localhost:8080* in your browser.
6. **Sign in** with your user account.
7. **Go to** *http://localhost:8080/api/jsonws*.

AVAILABLE JSON WEBSERVICES

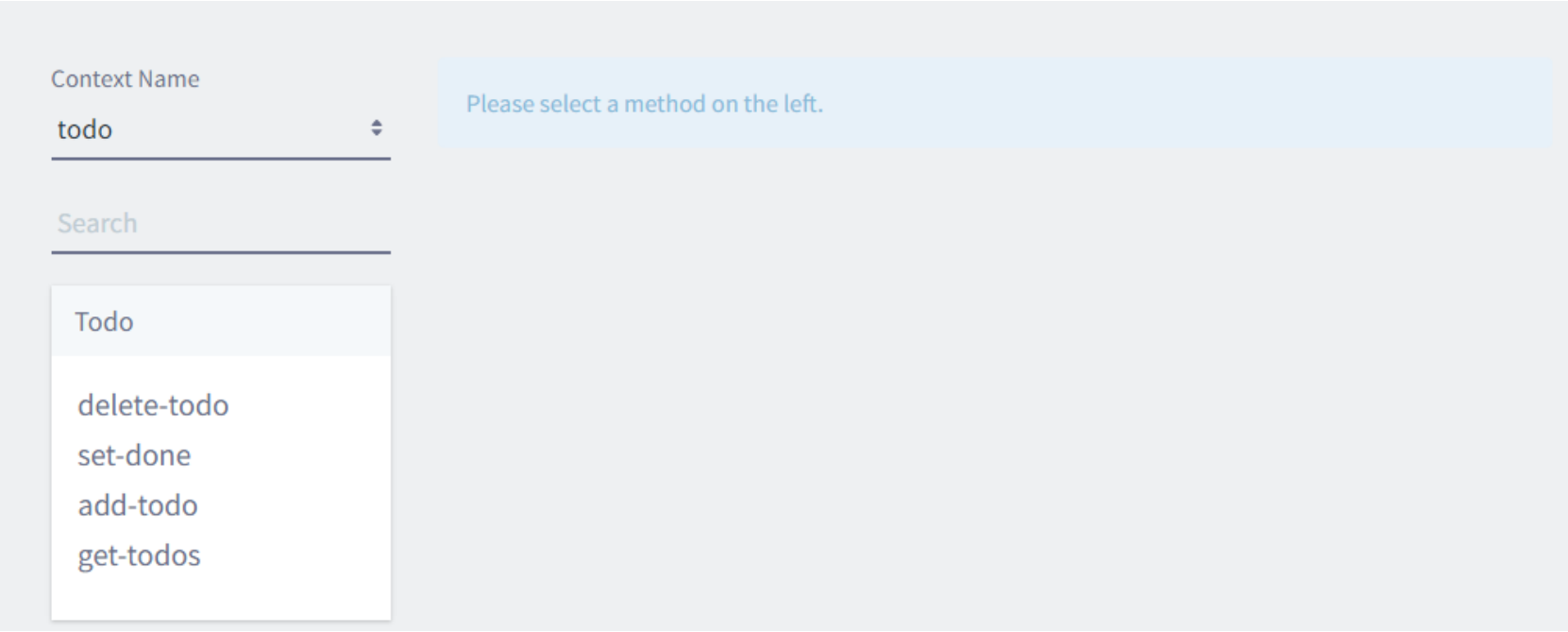
- Your browser should now show the list of JSON WebServices available in the platform.
- The back-end developers have also notified us that the services are exposed under the `todo` Context name.



EXERCISE: CHANGING THE CONTEXT

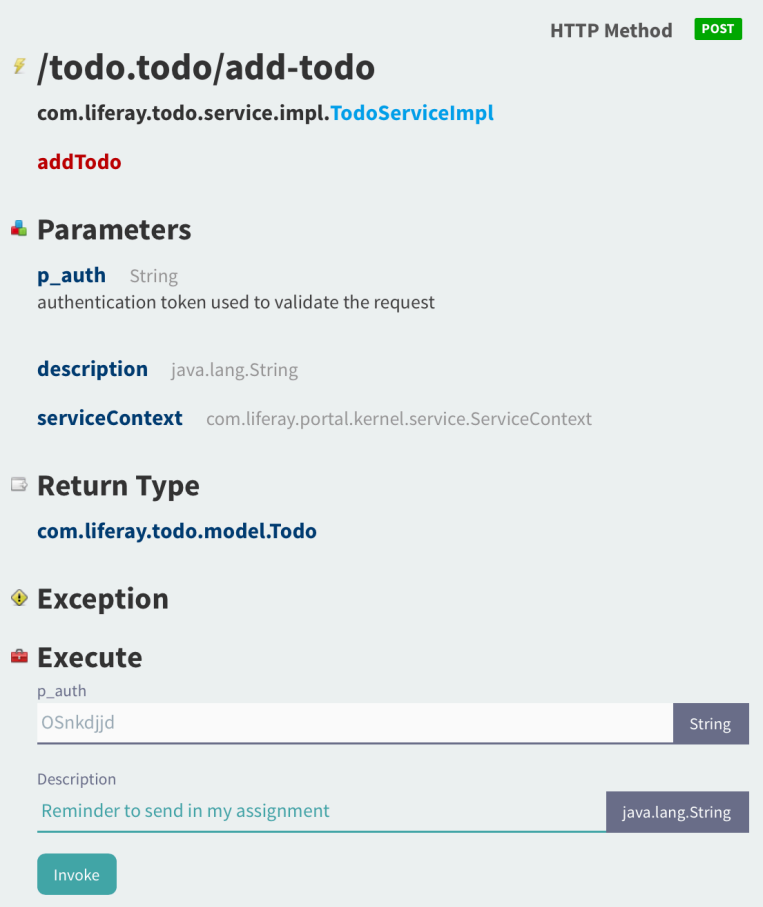
1. **Choose** *todo* from the *Context Name* drop-down.

✓ You should now see a list of service endpoints such as `get-todos`, `add-todos`, `set-done`, and `delete-todos`.



EXERCISE: SETTING A REMINDER

1. **Choose** `add-todo`.
2. **Type** *Reminder to send in my assignment* into the *Description* input field.
3. **Click** the *Invoke* button.



EXERCISE: USING JAVASCRIPT LATER

1. **Click** the `JavaScript` example tab to see an example of JavaScript invoking this service. We'll take note of this and use it in our code.

- *Bonus: You can play around with the other service endpoints to get a feel for what the API is offering.*

SOY TEMPLATES

- As mentioned previously, the back-end developers have already developed the SoyPortlet.
- The team has placed a list of `todo` into the Soy Template, which will allow us to access the data for display.

```
long groupId = themeDisplay.getScopeGroupId();
long userId = themeDisplay.getUserId();

List<Todo> todos = TodoLocalServiceUtil.getTodosByUserIdAndGroupId(userId,
groupId);

JSONSerializer jsonSerializer = JSONFactoryUtil.createJSONSerializer();
JSONDeserializer jsonDeserializer = JSONFactoryUtil.createJSONDeserializer();
List<Object> todoContainer = new ArrayList<>();
for (Todo todo : todos) {
  String json = jsonSerializer.serializeDeep(todo);
    todoContainer.add(jsonDeserializer.deserialize(json));
}
template.put("todos", todoContainer);
```

EXERCISE: SOY TEMPLATES

- As front-end developers, we are responsible for creating the following resources to render the UI:
  - **Todo.soy**: This file will be our primary Soy Template file.
  - **Todo.es.js**: This will be our Metal.js file.
  - **Todo.scss**: This will be our SCSS file to provide our styling.

1. **Open** *Brackets* if it's not already open.
2. **Click** on the dropdown in the left-hand side bar.
3. **Click** *Open Folder...*
4. **Go to** *exercises/front-end-developer-exercises/appendix-soy-templates*.
5. **Choose** the snippets folder.

- Let's start by laying out our Soy Template to provide the UI elements for our application.

EXERCISE: INPUT FIELDS

1. **Go to** *exercises/front-end-developer-exercises/appendix-soy-templates/src*.
2. **Open** the *Todo.soy* file in *Brackets*.
3. **Copy** the contents of the *01-input-fields* snippet.
4. **Replace** `<%-- Insert 01-input-fields ---%>` in the *Todo.soy* file with the *01-input-fields* snippet.
   - Format as needed.
   - Take note of the `data-onclick="addTodo"` attribute on our add button. We will provide the *onclick* method shortly.

- Next, we will create the UI elements to list the list of *Todos*.

EXERCISE: CREATING UI ELEMENTS

1. **Copy** the contents of the *02-list-todo* snippet.
2. **Replace** `<%-- Insert 02-list-todo ---%>` in *Todo.soy* with the *02-list-todo* snippet.
   - The snippet is taking advantage of the `{foreach}` and `{if}` commands to loop through the list of *todos* provided to the template and make UI decisions as to whether to mark the item as done.
   - We have also added another *onclick* method to the checkbox `data-onclick="finishTodo"`.

```
checked="true"
data-item-is-done="true"
```

EXERCISE: VIEWING COMPLETED TASKS

- For the final piece of our UI, we will provide a toggle to allow the User to view completed *todo* tasks.

1. **Copy** the contents of the *03-done-toggle* snippet.
2. **Replace** `<%-- Insert 03-done-toggle ---%>` in *Todo.soy* with the *03-done-toggle* snippet.
3. **Save** the file.
   - Once again, we have attached an *onclick* method to our toggle input, `data-onclick="toggleTodo"`, which we will implement in a moment.

- Now that we have our UI elements laid out in our Soy Template, review the various CSS classes we've added onto our UI elements.
- You can reference the Lexicon CSS modules for additional information about these CSS classes.

EXERCISE: METAL.JS

- With our UI now laid out in the Soy Template, let's provide the implementation of the *onclick* methods required by the UI.

1. **Open** the *Todo.es.js* file in *Brackets*.
   - Here, you will see that we have defined a *Todo* Metal.js component and registered our Soy Templates.

```
class Todo extends Component {
  ...
 // Register component
 Soy.register(Todo, templates);
 ...
}
```

EXERCISE: ADD TODO

1. **Copy** the contents of the *04-add-todo-method* snippet.
2. **Replace** `<%-- Insert 04-add-todo-method ---%>` in *Todo.es.js* with the *04-add-todo-method* snippet.
   - Take a look at the code. You can see that we're using the same JavaScript code to invoke the service as we saw previously when we were browsing the JSON service endpoints.

```
Liferay.Service(
  '/todo.todo/add-todo',
  {
    description: todoValue,
  },
....
);
```

EXERCISE: ADDING THE OTHER METHODS

- Let's repeat this process and add the other two methods.

1. **Copy** the contents of the *05-finishTodo-method* snippet.
2. **Replace** `<%-- Insert 05-finishtodo-method ---%>` in *Todo.es.js* with the *05-finishTodo-method* snippet.
   - Again, you can see that the code closely mirrors the JavaScript code from before.
3. **Copy** the contents of the *06-toggleTodo-method* snippet.
4. **Replace** `<%-- Insert 06-toggleTodo-method ---%>` in *Todo.es.js* with the *06-toggleTodo-method* snippet.
5. **Save** the file.

EXERCISE: CSS FOR THE APPLICATION

- Now that we have our UI and its JavaScript methods created using Soy Templates and Metal.js components, let's add the finishing touches and provide some CSS for our application.
- Then we can ship the completed code to the back-end developer and see our application in action.

1. **Open** the *Todo.scss* file in *Brackets*.
2. **Insert** the *07-todo-sass* snippet.
3. **Go to** *exercises/front-end-developer-exercises/appendix-soy-templates*.
4. **Copy** the *com.liferay.todo.web.jar*.
5. **Go to** your *[LIFERAY_HOME]* directory here: *liferay/bundles/liferay-dxp-digital-enterprise-[version]*.
6. **Paste** the JAR file into the */deploy* folder.

EXERCISE: VERIFYING THE APPLICATION

1. **Go to** *http://localhost:8080* in your browser.
2. **Sign in** with your User account.
3. **Open** the *Add* menu at the top right.
4. **Click** to open *Applications→Tools*.
5. **Drop** the *Todo Portlet* Application onto the page.

✓ Feel free to test the application's functionality by adding a new *Todo* and toggling the items.

VERIFYING THE APPLICATION

Todo Portlet

Here is the todo list

Add a todo

Buy some cheese

add

Reminder to send in my assignment

Show completed tasks

Notes: