JSON WEB SERVICES

JSON WEB SERVICES IN LIFERAY

- Liferay has many web services ready to use out of the box.
- These services include retrieving data and information about various assets, creating new assets, and even editing existing assets.
- To see a comprehensive list of the available web services, start up a bundle and navigate to *http://localhost:8080/api/jsonws*
- This list will include any custom web services that have been deployed to the bundle.
- These services are useful for creating single page applications, and can even be used to create custom front-ends both inside and outside of Liferay.

INVOKING WEB SERVICES VIA JAVASCRIPT

- In Liferay DXP, there is a global JavaScript object named `Liferay` that has many useful utilities.
- One method is `Liferay.Service`, which is used for invoking JSON web services.
- The `Service` method takes four possible arguments.

SERVICE METHOD ARGUMENTS

- **Required**
  1. `service {string|object}`: Either the service name, or an object with the keys as the service to call, and the value as the service configuration object
- **Optional**
  1. `data {object|node|string}`: The data to send to the service. If the object passed is the ID of a form or a form element, the form fields will be serialized and used as the data.
  2. `successCallback {function}`: A function to execute when the server returns a response. It receives a JSON object as its first parameter.
  3. `exceptionCallback {function}`: A function to execute when the response from the server contains a service exception. It receives the exception message as its first parameter.

SERVICE METHOD VS. STANDARD AJAX

- One of the major benefits of using the `Service` method vs. a standard ajax request is that it handles the authentication.

```
Liferay.Service(
    '/user/get-user-by-email-address',
    {
        companyId: Liferay.ThemeDisplay.getCompanyId(),
        emailAddress: 'test@liferay.com'
    },
    function(obj) {
        console.log(obj);
    }
);
```

- In this example, we are retrieving information about a user by passing in the `companyId` and `emailAddress` of the user in question.

RESPONSE DATA

- Response data resembles the following:

```
{
    "agreedToTermsOfUse": true,
    "comments": "",
    "companyId": "20116",
    "contactId": "20157",
    "createDate": 1471990639779,
    "defaultUser": false,
    "emailAddress": "test@liferay.com",
    "emailAddressVerified": true,
    "facebookId": "0",
    "failedLoginAttempts": 0,
    "firstName": "Test",
    ...
}
```

BATCHING REQUESTS

- Another format for invoking the `Service` method is by passing an object with the keys as the service to call, and the value as the service configuration object.

```
Liferay.Service(
    {
        '/user/get-user-by-email-address': {
            companyId: Liferay.ThemeDisplay.getCompanyId(),
            emailAddress: 'test@liferay.com'
        }
    },
    function(obj) {
        console.log(obj);
    }
);
```

INVOKING MULTIPLE SERVICES

- With this format, you can actually invoke multiple services with the same request by passing in an array of service objects.

```
Liferay.Service(
    [{'/user/get-user-by-email-address': {
                companyId: Liferay.ThemeDisplay.getCompanyId(),
                emailAddress: 'test@liferay.com'
            }},
        {
            '/role/get-user-roles': {
                userId: Liferay.ThemeDisplay.getUserId()
            }
        }
    ],
    function(obj) {
        // obj is now an array of response obejcts
        // obj[0] == /user/get-user-by-email-address data
        // obj[1] == /role/get-user-roles data
        console.log(obj);
    }
);
```

NESTING REQUESTS

- With nested service calls, you can bind information from related objects together in a JSON object.
- You can call other services within the same HTTP request and nest returned objects in a convenient way.
- You can use variables to reference objects returned from service calls. Variable names must start with a dollar sign, $.
- In this example, we will retrieve user data with `/user/get-user-by-id`, and use the contactId returned from that service to then invoke `/contact/get-contact` in the same request.

NESTING REQUESTS EXAMPLE

- You must flag parameters that take values from existing variables. To flag a parameter, insert the @ prefix before the parameter name.

```
Liferay.Service(
    {
        "$user = /user/get-user-by-id": {
            "userId": Liferay.ThemeDisplay.getUserId(),
            "$contact = /contact/get-contact": {
                "@contactId": "$user.contactId"
            }
        }
    },
    function(obj) {
        console.log(obj);
    }
);
```

NESTING REQUESTS RESULTS

- Here is what the response data would look like for that request.

```
{
    "agreedToTermsOfUse": true,
    "comments": "",
    "companyId": "20116",
    "contactId": "20157",
    "createDate": 1471990639779,
    "defaultUser": false,
    "emailAddress": "test@liferay.com",
    "emailAddressVerified": true,
    ...
    "contact": {
        "accountId": "20118",
        "birthday": 0,
        "classNameId": "20087",
        "classPK": "20156",
        "companyId": "20116",
        ...
    }
}
```

FILTERING RESULTS

- If you don't want all the properties returned by a service, you can define a whitelist of properties.
- Only the specific properties you request in the object are returned from your web service call.
- Here's how you whitelist the properties you need:

```
Liferay.Service(
    {
        '$user[emailAddress,firstName] = /user/get-user-by-id': {
            userId: Liferay.ThemeDisplay.getUserId()
        }
    },
    function(obj) {
        console.log(obj);
    }
);
```

SPECIFYING WHITELIST PROPERTIES

- To specify whitelist properties, you simply place the properties in square brackets (e.g., [whiteList]) immediately following the name of your variable.
- Here is what the filtered response looks like.

```
{
    "firstName": "Test",
    "emailAddress": "test@liferay.com"
}
```

INNER PARAMETERS

- When you pass in an object parameter, you'll often need to populate its inner parameters (i.e., fields).
- Consider a default parameter `serviceContext` of type `ServiceContext`.
- To make an appropriate call to JSON web services, you might need to set `serviceContext` fields such as `scopeGroupId`.

```
Liferay.Service(
    '/example/some-web-service',
    {
        serviceContext: {
            scopeGroupId: 123
        }
    },
    function(obj) {
        console.log(obj);
    }
);
```

Notes: