



# Exercises for **Mastering Liferay Client Extensions**

SKO Technical Clinic



# Exercises for Mastering Liferay Client Extensions - SKO 2025 Edition

---

## Table of Contents

- [Exercise 1: Setting Up the SKO Workspace](#)
- [Exercise 2: Exporting the Contact Us Object Definition](#)
- [Exercise 3a: Preparing Clarity's Distributor Management App Payload](#)
- [Exercise 3b: Configuring the Batch Client Extension](#)
- [Exercise 3c: Deploying the Client Extension](#)
- [Exercise 4: Deploying Clarity's Ticket List Custom Element](#)
- [Exercise 5: Updating Clarity's Frontend Tokens](#)
- [Exercise 6a: Creating an Accessibility Menu with a Global JS Client Extension](#)
- [Exercise 6b: Applying the Global JS Client Extension to Clarity's Home Page](#)

## Exercise 1: Setting Up the SKO Workspace

Throughout the technical clinic, you'll use a local Liferay workspace for the hands-on exercises and practice what you learn. For that purpose, ensure you've completed the SKO Technical Clinic Prerequisites sent via email.

1. Open your terminal and run this command to verify Git is installed:

```
git version
```

**Note:** If you're on Windows, use Command Prompt, PowerShell, or BASH to execute terminal commands.

This returns the version of your git installation. For example,

```
git version 2.45.2
```

If the Git command isn't found, please see official documentation for how to install Git for your OS ([macOS](#) | [Windows](#) | [Linux/Unix](#)).

2. Verify Java JDK 21 is installed:

```
java -version
```

The JDK version should display:

```
openjdk version "21.0.5" 2024-10-15 LTS
OpenJDK Runtime Environment Zulu21.38+21-CA (build 21.0.5+11-LTS)
OpenJDK 64-Bit Server VM Zulu21.38+21-CA (build 21.0.5+11-LTS, mixed
mode, sharing)
```

If Java isn't installed, you can find the appropriate OpenJDK distribution installer for your OS [here](#). Alternatively, you can download the JDK as a ZIP (Windows) or TAR.GZ (Linux/Mac) package. To install, extract the file in a folder of your choice, then set the JAVA\_HOME environment variable to that folder.

**Note:** If you support multiple Liferay projects and need to switch between different JDK versions, consider using a Version Manager:

- **Unix-based systems:**
  - [SDKMAN!](#)
  - [jEnv](#)
- **Windows:**
  - [Jabba](#)
  - [JVMS](#)

### 3. (Optional) Verify Blade is installed:

```
blade version
```

It should return the CLI's version:

```
blade version 6.0.0.202404102137
```

If Blade isn't installed, see [Blade CLI](#) installation instructions.

If the output indicates there's a newer version, run this command to update it:

```
blade update
```

**Note:** While we recommend using Blade to set up Liferay Workspace, you can also use Gradle to complete the process manually. See [Creating a Liferay Workspace Manually](#) for more information.

### 4. In your terminal, go to your desired folder and clone the training workspace to your computer:

```
git clone https://github.com/liferay/sko-2025
```

This saves a copy of the project in your current terminal directory.

**Note:** If you've cloned the repo previously, ensure your workspace is up to date by running `git pull origin main`.

5. Go to the workspace's root folder in your terminal:

```
cd sko-2025/
```

6. Initialize your Liferay bundle:

```
blade server init
```

This downloads and builds dependencies for running Liferay, including the Liferay server.

7. Use Blade to start your Liferay server:

```
blade server run
```

Alternatively,

**Unix-based:**

```
./bundles/tomcat/bin/catalina.sh run
```

**Windows:**

```
.\bundles\tomcat\bin\catalina.bat run
```

**Tip:** Wait until you see `org.apache.catalina.startup.Catalina.start Server startup in X milliseconds` to indicate startup completion.

8. When finished, access your Liferay DXP instance by going to `http://localhost:8080/` in your browser.
9. Sign in using these credentials:
  - Username: `admin@clarityvisionsolutions.com`
  - Password: `learn`



10. Take some time to explore the site and resources included in the training workspace.

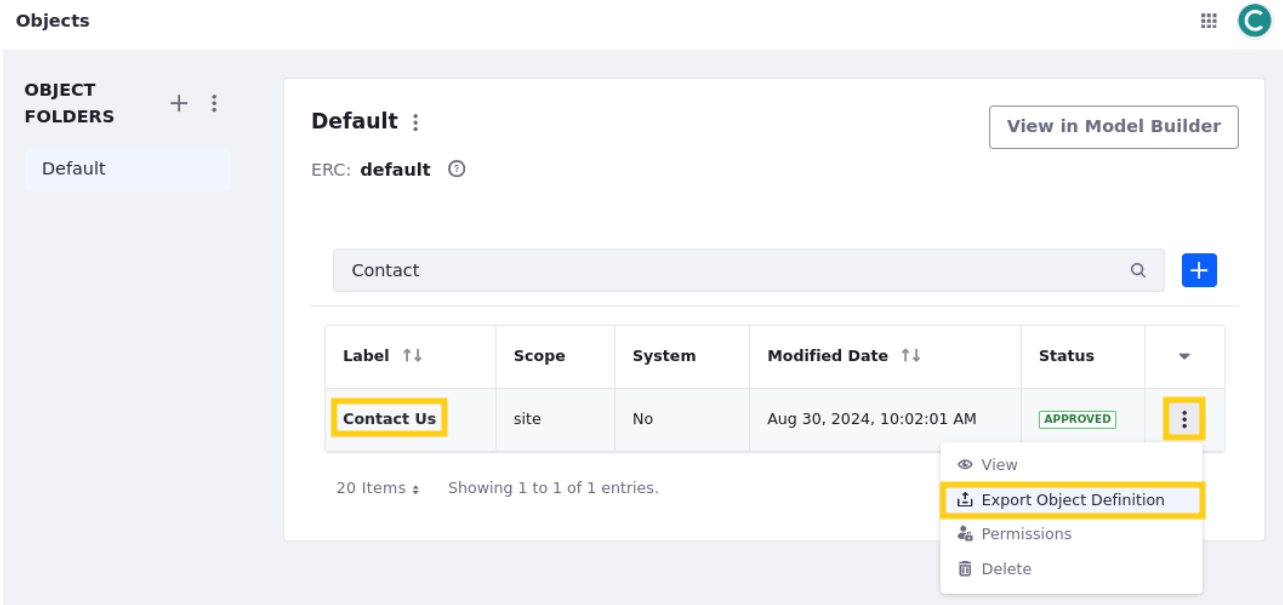
Great! With your environment set up, you're ready to start contributing to Clarity's solutions!

## Exercise 2: Exporting the Contact Us Object Definition

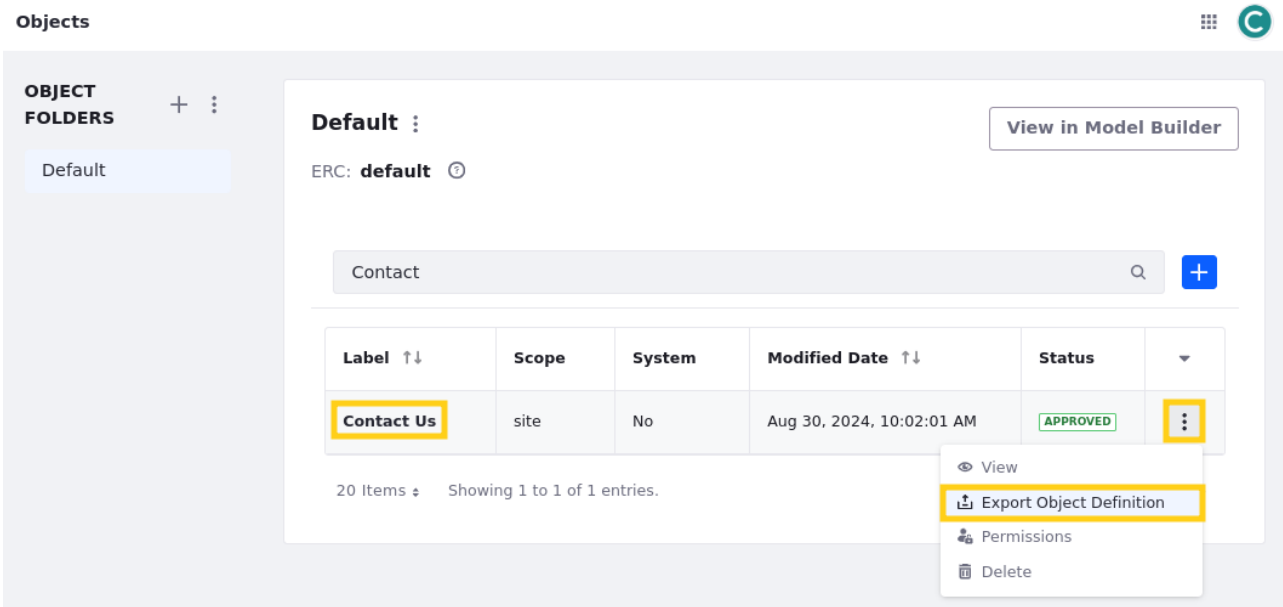
Here, you'll export Clarity's Contact Us object definition and explore its associated JSON file.

1. In your running Liferay instance, sign in as the Clarity Admin user.
- Username: `admin@clarityvisionsolutions.com`

◦ Password: `learn`
2. Open the *Global Menu* (  ), go to the *Control Panel* tab, and click *Objects*.
3. Click *Actions* (  ) for the Contact Us object and select *Export Object Definition*.



4. Click on the *Contact Us* object and navigate to the *Fields* tab, identifying the currently included fields.



5. Open the downloaded `Object_Definition_ContactUs_[...].json` file in a text editor or IDE.
- Note:** Many text editors and IDEs offer extensions to “prettify” JSON code, improving its readability for human comprehension.
6. Examine the file's JSON elements and nested values.

Great! By successfully exporting one of Clarity's object definitions and exploring its JSON structure, you've completed the crucial first steps for preparing a batch client extension. Next, you'll learn how to package exported files from Clarity's Distributor Management app into a batch client extension.

## Exercise 3a: Preparing Clarity's Distributor Management App Payload

Here, you'll package the Distributor Management app's exported resources into a client extension project and create a batch payload from the object definition file.

1. Open a file explorer and navigate to the [exercises/exercise-3/](#) folder in your course workspace.
2. Rename the [liferay-sample-batch](#) folder to [clarity-distributor-mgmt-batch](#).

**Note:** The [liferay-sample-batch](#) client extension was downloaded from the [Liferay Sample Workspace](#). As a best practice, use examples within this workspace as the baseline for your own client extension projects, as this serves as the primary source of truth for client extension implementation.

3. Within the [clarity-distributor-mgmt-batch/batch/](#) folder, delete the existing [.json](#) files.
4. From the previous [exercise-3/](#) folder, move these files into the [clarity-distributor-mgmt-batch/batch/](#) folder:

- [00-list-type-definition.batch-engine-data.json](#)
- [02-user-role.batch-engine-data.json](#)
- [03-workflow-definition.batch-engine-data.json](#)
- [04-notification-definition.batch-engine-data.json](#)
- [Object\\_Definitions.json](#)

These files contain all the resources for Clarity's Distributor Management app: the picklists, user roles, workflow, notification templates, and the object definitions.

**Note:** It's best practice to include a numeric prefix to each file name to determine the order in which they're imported upon deployment. This is useful when subsequent files require pre-populated dependencies from other files.

5. Navigate to the [clarity-distributor-mgmt-batch/batch/](#) folder.
6. Rename the [Object\\_Definitions.json](#) file to [01-object-definition.batch-engine-data.json](#).

This puts the object definition batch file in the second deployment position.

7. From the [exercise-3/code-samples/](#) folder, open the [object-payload-configuration.txt](#) file and copy its content.

This file contains the payload configuration block for the object definitions.

8. Open the [clarity-distributor-mgmt-batch/batch/01-object-definition.batch-engine-data.json](#) file with a text editor or IDE.
9. Paste the code snippet from the [object-payload-configuration.txt](#) file within the first opening curly brace (`{`), prior to the [items](#) block:

This defines the batch payload's configuration and specifies the object definitions as the data block.

10. Your file should resemble this:

```
{
  "configuration": {
    "className":
    "com.liferay.object.admin.rest.dto.v1_0.ObjectDefinition",
    "parameters": {
      "containsHeaders": "true",
      "createStrategy": "UPSERT",
      "onErrorFail": "ON_ERROR_FAIL",
      "updateStrategy": "UPDATE"
    },
    "taskItemDelegateName": "DEFAULT"
  },
  "items": [
    {
      "active": true,
      "defaultLanguageId": "en_US",
      "enableCategorization": true,
      "enableIndexSearch": true,
      "enableObjectEntryDraft": true,
      "externalReferenceCode": "D4B8_DISTRIBUTOR_APPLICATION",
      [...]
      "titleObjectName": "creator"
    }
  ]
}
```

**Note:** Ensure the object definitions are under the `items` block as a valid JSON before proceeding.

11. Save the file.

Great! You've moved the Distributor Management app's resources into a client extension project and created a batch payload from the object definition file. Next, you'll define the `client-extension.yaml` file.

## Exercise 3b: Configuring the Batch Client Extension

Here, you'll define the structure, resources, and configurations needed to deploy and manage the batch client extension.

1. Within the `clarity-distributor-mgmt-batch/` project folder, open the `client-extension.yaml` with a text editor or IDE.
2. Delete the file's existent content.
3. From the `exercise-3/code-samples/` folder, open the `client-extension-assembly-block.txt` file and copy its content.

4. Paste the code snippet in the `client-extension.yaml` file you opened previously.


This adds the `assemble` block to specify which resources the client extension should package during the build process.

5. Open the `client-extension-definition-block.txt` file in the `code-samples/` folder, copy the code snippet, and paste it in the `client-extension.yaml` file under the `assemble` block.

This adds the batch client extension definition for Clarity's Distributor Management app, including its name, the OAuth 2.0 server reference, and type.

6. Open the `client-extension-server-block.txt` file in the `code-samples/` folder, copy the code snippet, and paste it in the `client-extension.yaml` file under the client extension definition block.

This adds an OAuth 2.0 headless server client extension for authorizing API calls with the necessary scopes for the batch client extension.

**Note:** To find the correct API scopes for your batch client extension, go into your Liferay instance's UI, open the *Global Menu* () , go to the *Control Panel* tab, and click *OAuth 2 Administration*. Select an OAuth 2.0 application from the list and go to the *Scopes* tab. This section displays all available Liferay API scopes.

7. Your file should resemble this:

```
assemble:
  - from: batch
    into: batch
clarity-distributor-mgmt-batch:
  name: Clarity Distributor Management Batch
  oAuthApplicationHeadlessServer: clarity-distributor-mgmt-batch-
  oauth-application-headless-server
  type: batch
clarity-distributor-mgmt-batch-oauth-application-headless-server:
  .serviceAddress: localhost:8080
  .serviceScheme: http
  name: Clarity Distributor Management Batch OAuth Application
  Headless Server
  scopes:
    - Liferay.Headless.Admin.List.Type.everything
    - Liferay.Headless.Admin.User.everything
    - Liferay.Headless.Admin.Workflow.everything
    - Liferay.Headless.Batch.Engine.everything
    - Liferay.Notification.REST.everything
    - Liferay.Object.Admin.REST.everything
  type: oAuthApplicationHeadlessServer
```

8. Save the file.

With the client extension set up, you can now move it to the appropriate workspace location.



9. Move the `clarity-distributor-mgmt-batch/` project folder into the `client-extensions/` folder of your course workspace.

**Note:** Copying and pasting the project will result in a deployment failure due to the duplicate client extension folders. To prevent this, move the project to the `client-extensions/` folder.

Great! You've fully configured Clarity's Distributor Management batch client extension. Next, you'll deploy it into your Liferay environment.

## Exercise 3c: Deploying the Client Extension

Here, you'll deploy the batch client extension to add the Distributor Management app into your Liferay instance.

1. Open a terminal and navigate to the `client-extensions/clarity-distributor-mgmt-batch/` in your course workspace.
2. Run this command to build and deploy the client extension:

```
blade gw clean deploy
```

In your Liferay logs, you'll find various messages related to import tasks executed by the `BatchEngineImportTaskExecutorImpl` module. These import tasks correspond to the files within the `batch/` folder.

3. Open your instance logs and search for a message similar to this:

```
[...] Started batch engine import task 904
```

This informs you that the batch engine has started an import task with the assigned ID `904`.

4. Search for another message similar to this:

```
[...] Finished batch engine import task 904 in 48ms
```

This indicates that the import task with the ID `904` has finished.

**Note:** You can use the import task ID (e.g., `904`) to retrieve information from the Batch API for troubleshooting errors and unexpected behaviors. Explore this in more detail in the Mastering Liferay's Headless APIs (*Coming Soon*) course.

5. Verify it deploys successfully.


```
INFO [fileinstall-directory-watcher][BundleStartStopLogger:68] STARTED  
claritydistributormgmtbatch_7.4.13 [1462]
```

---

Now that you've deployed the batch client extension, you can examine the Distributor Management app.


6. In your Liferay instance, sign in as the Clarity Admin user.

- Username: [admin@clarityvisionsolutions.com](mailto:admin@clarityvisionsolutions.com)
- Password: [learn](#)

7. Open the *Global Menu* () , go to the *Control Panel* tab, and click *Objects*.

8. Verify these objects are present:

- Distributor Application
- Application Evaluation

9. In the Global Menu () , go to the *Control Panel* tab and click *Picklists*.


10. Verify these picklists are present:

- Annual Purchase Volumes
- Application States
- Assessment Scores
- Decisions
- Distribution Channels
- Distribution Regions
- Product Types
- Recommendations

11. In the Global Menu () , go to the *Control Panel* tab and click *Templates* under Notifications.

12. Verify these notification templates are present:

- Application Approved, Applicant, Email
- Application Denied, Applicant, Email
- Application Received, Applicant, Email
- Distributor Application Submitted, Admin, User

13. In the Global Menu () , go to the *Control Panel* tab and click *Roles*.

14. Verify these user roles are present:

- Business Development Manager
- Business Development Specialist

15. In the Global Menu () , go to the *Applications* tab and click *Process Builder*.

16. Verify the Distributor Manager Approval workflow is present.

Great! You've deployed the batch client extension and explored the Distributor Management app's content. Next, you'll deploy a user interface for Clarity's Ticketing app.

## Exercise 4: Deploying Clarity's Ticket List Custom Element

Here, you'll explore and deploy a React application developed by Clarity's team as a Custom Element Client Extension, designed to retrieve, filter, and display ticket data.

1. Open a file explorer and navigate to the `exercises/exercise-4/` folder in your course workspace.
2. Rename the `react-app/` folder to `clarity-ticketing-ui`.
3. Within the `clarity-ticketing-ui/` folder, open the `webpack.config.js` file and paste the following code to the output block.

```
output: {
  clean: true,
  environment: {
    dynamicImport: true,
    module: true,
  },
  filename: WEBPACK_SERVE ? '[name].js' : '[name].[contenthash].js',
  library: {
    type: 'module',
  },
  path: path.resolve('build', 'static'),
},
```

This sets the `library` format, which specifies how the output bundle should be exposed.

4. From the `clarity-ticketing-ui/public` folder, open the `index.html` file in a text editor or IDE.
5. Replace the `<root>` and `<section>` tags with `<clarity-ticketing-ui>`, the name of our custom element. For example,

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
  <title>Clarity Ticketing UI</title>
</head>
<body>
  <clarity-ticketing-ui></clarity-ticketing-ui>
  <script type="module" src="build/static/index.js"></script>
</body>
</html>
```

6. From the `clarity-ticketing-ui/` folder, open the `index.js` file in a text editor or IDE and replace it with the following code.

```

import React, { useState, useEffect } from 'react';
import ReactDOM, { render } from 'react-dom';
import './assets/style.css';
import TicketsList from './assets/components/TicketsList';
import { HashRouter, Route, Routes } from "react-router-dom";
import App from './App';

// Custom Element class
class CustomElement extends HTMLElement {
  connectedCallback() {
    // Ensure the React component is rendered only once
    if (!this.rendered) {
      // Create a container if it doesn't exist
      const container = document.createElement('div');
      container.id = 'tickets-root';
      this.appendChild(container);
      // Render the React component into the container
      ReactDOM.render(<TicketsList />, container);
      this.rendered = true;
    }
  }
  disconnectedCallback() {
    // Clean up the React component when the element is removed
    const container = this.querySelector('#tickets-root');
    if (container) {
      ReactDOM.unmountComponentAtNode(container);
    }
    this.rendered = false;
  }
}
// Define the custom element
const ELEMENT_NAME = 'clarity-ticketing-ui';
if (!customElements.get(ELEMENT_NAME)) {
  customElements.define(ELEMENT_NAME, CustomElement);
}
// Automatically add the custom element to the page if not already
present
document.addEventListener('DOMContentLoaded', () => {
  if (!document.querySelector(ELEMENT_NAME)) {
    const customElement = document.createElement(ELEMENT_NAME);
    document.body.appendChild(customElement);
  }
});

```

This replaces the default use of `render()` on the `ticket-root` div, leveraging a Web Component to define the React app as a reusable and self-contained custom element.

7. Within the `clarity-ticketing-ui/` folder, create a `client-extension.yaml` file and add the following code to it.

```
assemble:
  - from: build/static
    into: static
clarity-ticketing-ui:
  friendlyURLMapping: clarity-ticketing-ui
  htmlElementName: clarity-ticketing-ui
  instanceable: false
  name: Clarity Ticketing UI
  portletCategoryName: category.client-extensions
  type: customElement
  urls:
    - index.*.js
  useESM: true
```



8. Move the `clarity-ticketing-ui/` folder into the `client-extensions/` folder of your course workspace.
9. Open a terminal and navigate to the `client-extensions/clarity-ticketing-ui/` folder.
10. Run this command to build and deploy the custom element client extension:

```
blade gw clean deploy
```

11. Verify that the client extension deploys successfully:

```
2025-01-28 11:50:59.076 INFO [fileinstall-directory-watcher]
[BundleStartStopLogger:68] STARTED clarityticketingui_7.4.13 [1462]
```

Now that you've deployed the custom element client extension, you can examine the Ticketing app UI.

12. In your Liferay instance, sign in as the Clarity Admin user.
  - Username: `admin@clarityvisionsolutions.com`
  - Password: `learn`
13. Open the *Site Menu* () , click *Page Tree*, and select the *Tickets* page.
14. Click *Edit* () to start editing the page.
15. In the Fragments and Widgets search bar, search for `Clarity Ticketing UI`.
16. Drag and drop the *Clarity Ticketing UI* widget to the page.
17. Click *Publish*.

Great! You've successfully deployed a custom element client extension for retrieving and displaying Clarity's ticket data. Next, you'll modify Clarity's current theme CSS client extension.

## Exercise 5: Updating Clarity's Frontend Tokens

Here, you'll add a new frontend token definition to Clarity's current Theme CSS client extension. You'll also import a new font family to override the current one.

1. Open a file explorer and navigate to the [exercises/exercise-5/](#) folder in your course workspace.
2. Open the [frontend-token-definition.json](#) file and examine its contents.
3. Move the [frontend-token-definition.json](#) file to the [client-extensions/clarity-theme/src/](#) folder in your course workspace.
4. From the [clarity-theme/](#) folder, open the [client-extension.yaml](#) file in a text editor or IDE.
5. Add this line above the name property:

```
frontendTokenDefinitionJSON: src/frontend-token-definition.json
```

For example,

```
assemble:
  - from: build/buildTheme/img
    into: static/img
  - from: assets
    into: static
clarity-theme:
  clayURL: css/clay.css
  mainURL: css/main.css
  frontendTokenDefinitionJSON: src/frontend-token-definition.json
  name: Clarity Theme CSS
  type: themeCSS
```

6. From the [client-extensions/clarity-theme/src/css/dialect/variables/](#) folder, open the [\\_color\\_scheme.scss](#) file and update the following variables by replacing their values with the following frontend token definition variables.

```
--color-action-primary-default: var(--btn-primary-bg),
--color-action-primary-hover: var(--btn-primary-bg),
--color-action-primary-inverted: var(--btn-primary-color),
--color-brand-primary: var(--card-category-color),
--color-neutral-10: var(--card-title-color),
```

This change maps the theme's variables to the corresponding tokens in the [frontend-token-definitions.json](#) file.

7. From the [client-extensions/clarity-theme/src/css/](#) folder, open the [\\_import.scss](#) file and append this snippet:

```
@import url('https://fonts.googleapis.com/css2?
family=Roboto+Slab:wght@100..900&display=swap');
* {
  font-family: 'Roboto Slab', serif;
}
```


This imports a font family from an external URL to override the current font.

8. Save your changes.
9. Open a terminal and navigate to the `client-extensions/clarity-theme/` project folder in your course workspace.
10. Run this command to build and deploy the client extension:

```
blade gw clean deploy
```

11. Verify it deploys successfully.

```
2025-01-24 14:08:34.676 INFO [fileinstall-directory-watcher]
[BundleStartStopLogger:68] STARTED claritytheme_7.4.13 [1463]
```

12. In your Liferay instance, open the Site Menu () , expand Design, and click Style Books.
13. Select *Clarity Kids Style Book* to start editing it.
14. In the dropdown menu, select *clarity-kids-components* and verify that the new tokens are available.
15. Modify the colors using the available tokens and observe the real-time changes on your page.

This token definition belongs to the theme set for pages.

Theme: Dialect

clarity-kids-components

## BUTTONS

### Primary Button Background Color



#0B5FFF



### Button Text Color



#FFFFFF



## CARDS

### Card Title Color



#272833



### Card Category Color



#F07D0A



16. Return to the Home page and confirm that it stays with the same style.

17. Also, confirm that the fonts were changed after you deployed the client extension.

Great! You've used the theme CSS client extension to add new frontend tokens to Clarity's style book. The client extension approach enables you to create distinct visual identities for sub-brands quickly and efficiently. Next, you'll learn more about implementing custom functionality with Global JS client extensions.

## Exercise 6a: Creating an Accessibility Menu with a Global JS Client Extension

Here, you'll create and deploy an accessibility menu using a Global JS client extension.

1. Open a file explorer and navigate to the [exercises/exercise-6/code-samples/](#) folder in your course workspace.
2. Open the [01-assemble-block.txt](#) file with a text editor or IDE, and examine its content.

```
- from: build/static  
  into: static
```



This file contains path properties for the `assemble` block in a `client-extension.yaml` file. The block specifies which resources the client extension should package during the build process and their output location.

3. Copy this code snippet.
4. Go to the `client-extensions/clarify-theme/` folder in your course workspace and open the `client-extension.yaml` file.
5. Paste the code from the `01-assemble-block.txt` file at the bottom of the `assemble` block. For example,

```
assemble:
  - from: build/buildTheme/img
    into: static/img
  - from: assets
    into: static
  - from: build/static
    into: static
clarify-theme:
[...]
```

**Note:** The order of the properties within the `assemble` block does not affect the configuration. However, ensure that the indentation matches the rest of the content.

6. From the `exercise-6/code-samples/` folder, open the `02-definition-block.txt` file and examine its content.

```
clarity-global-js:
  name: Clarity Global JS
  type: globalJS
  url: global.*.js
```

This file contains the client extension definition block, specifying its name, type, and source URL for the build process.

7. Copy this code snippet.
8. Within the `client-extensions/clarify-theme/client-extension.yaml` file, paste the code after the `clarity-theme-favicon-light` definition block. For example,

```
[...]
clarity-theme-favicon-light:
  name: Clarity Theme Favicon Light
  type: themeFavicon
  url: clarity-favicon-light.svg
clarity-global-js:
```

```
name: Clarity Global JS
type: globalJS
url: global.*.js
```

9. From the `exercise-6/assets/` folder, move the `global.js` file into the `client-extensions/clarity-theme/assets/` folder.
10. Move the `webpack.config.js` file from the `exercise-6/` folder to `client-extensions/clarity-theme/`.
11. Rename the `03-package.json` file from the `exercise-2/` folder to `package.json`.
12. Replace the current `package.json` file in the `client-extensions/clarity-theme/` folder with the file from the previous step.
13. Open a terminal and navigate to the `client-extensions/clarity-theme/` folder in your course workspace.
14. Run this command to build and deploy the client extension:

```
blade gw clean deploy
```




15. Verify it deploys successfully.

```
2025-01-24 14:08:34.676 INFO [fileinstall-directory-watcher]
[BundleStartStopLogger:68] STARTED claritytheme_7.4.13 [1463]
```

Now that you've included the Global JS client extension in `clarity-theme` and redeployed it, you can implement the accessibility menu on Clarity's website.

## Exercise 6b: Applying the Global JS Client Extension to Clarity's Home Page

Here, you'll apply the Global JS client extension to Clarity's home page.

1. In your running Liferay instance, click *Edit* () to start editing the Home page.
2. Click *Page Design Options* () on the left menu.
3. Click *More Page Design Options* ()
4. On the Design tab, under the Customization section, click the *JavaScript* tab.
5. Click *Add JavaScript Client Extensions* and select *In Page Head* in the dropdown.

General

**Design**

SEO

Open Graph

Custom Meta Tags

## Design

### Basic Settings

**Favicon**

Clarity Theme Favicon

**Master**

Primary Master Page

**Style Book**

Styles from Theme

This icon will be shown in the navigation menu.

**Icon**

Default

### Customization

CSS JavaScript

#### JAVASCRIPT CLIENT EXTENSIONS

+ Add JavaScript Client Extensions

In Page Head were loaded.

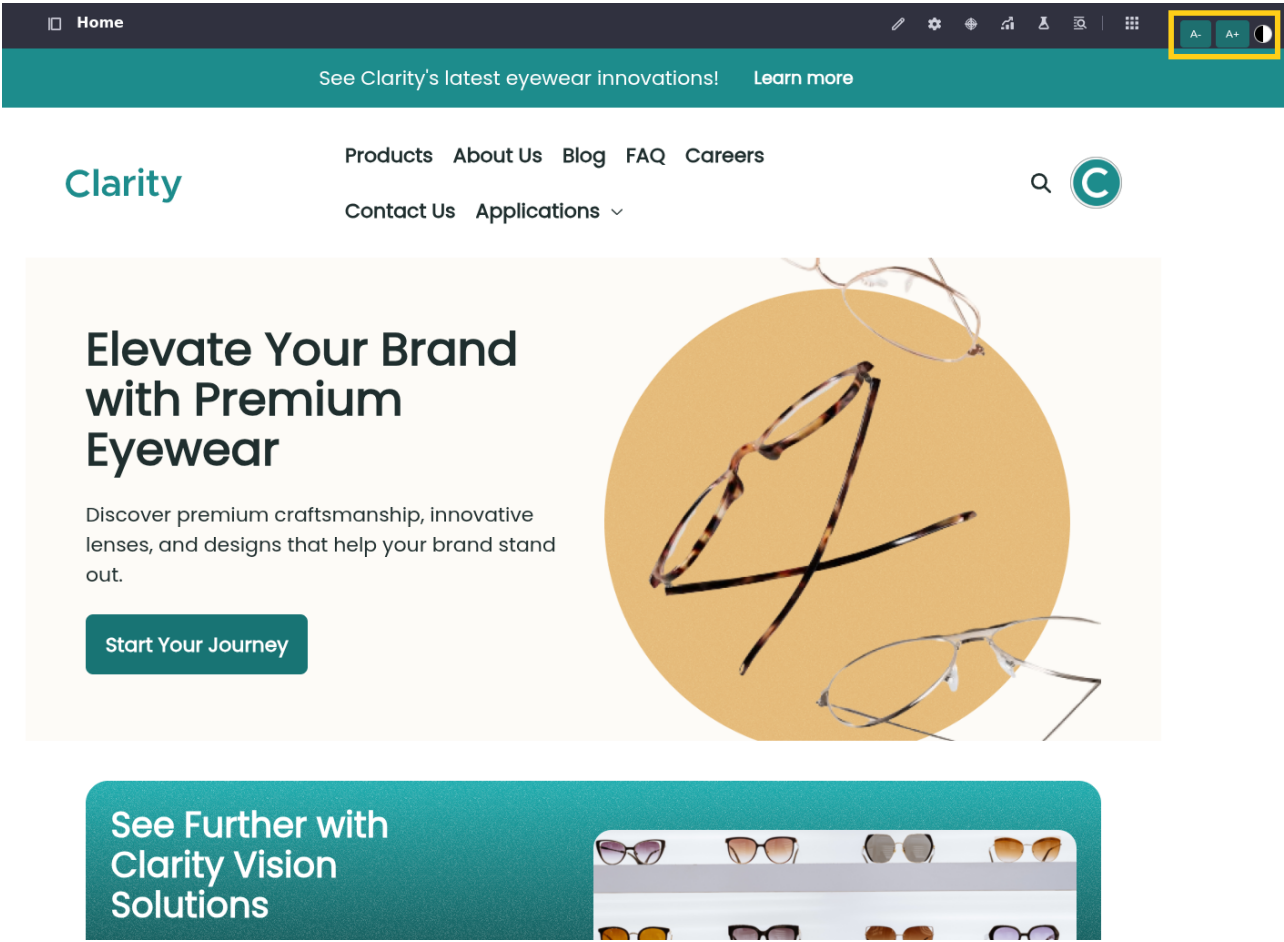
In Page Bottom

#### CUSTOM JAVASCRIPT

JavaScript

JavaScript

6. Select the *Clarity Global JS* checkbox and click *Add*.
7. Scroll to the bottom of the page and click *Save*.
8. Return to the Home page and publish it.
9. Click the *A+* button in the top-right corner of the page to increase the font size.



Great! Now that you've added the Global JS Client Extension and implemented Clarity's accessibility menu to their Home page, you can control the page's font size and apply a grayscale filter.