All Types of Client Extensions in Liferay 7.4

Frontend, Configuration, Microservice, and Batch Client Extensions

Liferay DXP Technical Notes

September 28, 2026

Abstract

This document provides a structured explanation of Liferay DXP 7.4 Client Extensions. It introduces the four major classifications: Frontend, Configuration, Microservice, and Batch Client Extensions. It also explains common subtypes, client-extension.yaml, Liferay Workspace structure, deployment, OAuth 2 integration, external microservices, frontend applications, configuration as code, batch data deployment, security considerations, and architectural selection guidelines.

Contents

1 Introduction 4

2 Why Client Extensions? 4

3 The Four Major Classifications 5

4 Classification Summary 5

I Frontend Client Extensions 5

5 Frontend Client Extensions 5

6 CSS Client Extension 6

7 JavaScript Client Extension 7

8 Theme CSS Client Extension 7

9 Theme Favicon Client Extension 8

10 Custom Element Client Extension 8

11 iFrame Client Extension 9

12 Custom Element vs iFrame 9

II Configuration Client Extensions 9

13 Configuration Client Extensions 9

14 Why Configuration as Code? 10

15 OAuth Configuration Client Extensions 10

16 OAuth User Agent 10

17 OAuth Headless Server 11

18 Instance Settings Client Extension 11

19 Language Client Extension 11

III Microservice Client Extensions 11

20 Microservice Client Extensions 11

21 Microservice Architecture 12

22 Object Action Client Extension 12

23 Workflow Action Client Extension 13

24 Workflow Condition Client Extension 13

25 Notification Client Extension 14

26 Custom Microservice 14

IV Batch Client Extensions 15

27 Batch Client Extensions 15

28 Batch Architecture 15

29 Batch Engine Data 15

30 When to Use Batch Client Extensions 16

31 client-extension.yaml 16

32 Liferay Workspace Structure 16

33 Packaging 17

34 LUFFA 17

35 Build and Deployment 17

36 Frontend Packaging 18

37 Microservice Packaging 18

38 OAuth 2 and Client Extensions 19

39 Client Extensions vs OSGi Modules 19

40 When Should OSGi Still Be Used? 20

41 How to Choose the Right Type 20

42 Example Composite Application **21**

43 Example Quantum Application **21**

44 Security Considerations **22**

45 Advantages **23**
45.1 Loose Coupling . . . 23
45.2 Technology Freedom . . . 23
45.3 Independent Scaling . . . 23
45.4 Upgrade Resilience . . . 23
45.5 Cloud Readiness . . . 23

46 Limitations **23**

47 Troubleshooting **24**
47.1 Client Extension Does Not Appear . . . 24
47.2 Frontend Resource Not Loaded . . . 24
47.3 401 Unauthorized . . . 24
47.4 403 Forbidden . . . 24
47.5 Microservice Not Reachable . . . 24

48 Interview Questions and Answers **25**

49 Multiple Choice Questions **25**

50 Quick Revision Table **29**

51 The Big Picture **29**

52 Key Takeaway **30**

53 Version Considerations **31**

54 Official Reference **31**

1 Introduction

Liferay Client Extensions provide a modern mechanism for extending Liferay DXP without deploying application code directly into the Liferay JVM.

Traditionally, Liferay applications frequently relied on:

- OSGi modules
- MVC Portlets
- Service Builder
- REST Builder
- JSP customizations
- Traditional themes

These remain relevant for certain use cases, but they execute inside or are tightly associated with the Liferay runtime.

Client Extensions provide a more loosely coupled architecture.

$$\boxed{\text{Client Extension} \longleftrightarrow \text{Liferay APIs / Extension Points}}$$

Core Idea

Client Extensions allow developers to customize Liferay's frontend, configuration, data, and external business logic while minimizing direct dependencies on Liferay's internal implementation.

2 Why Client Extensions?

Client Extensions can provide several architectural benefits:

- Reduced coupling to Liferay internals
- Easier upgrades
- Independent technology choices
- External microservice integration
- Reusable configuration
- Modern frontend integration
- Better separation of concerns
- Independent service scaling
- Cloud-friendly deployment

The general principle is:

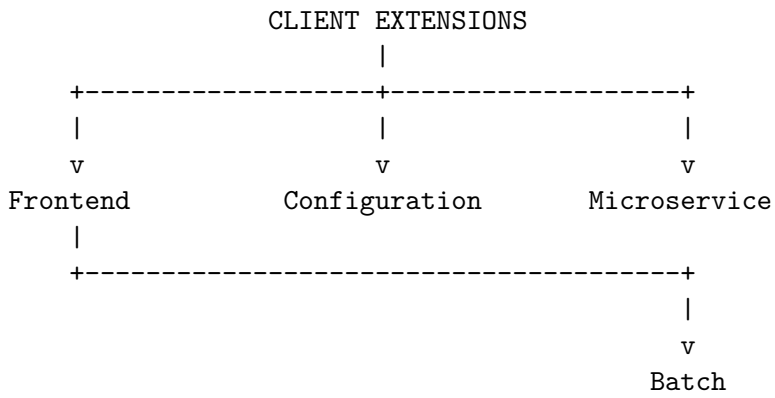
$$\boxed{\text{Extend Liferay} \neq \text{Modify Liferay Core}}$$

3 The Four Major Classifications

Liferay Client Extensions are organized into four major classifications:

1. Frontend Client Extensions
2. Configuration Client Extensions
3. Microservice Client Extensions
4. Batch Client Extensions

Conceptually:



Another useful way of thinking about them is:

$$\boxed{\text{Client Extensions} = \text{UI} + \text{Configuration} + \text{External Logic} + \text{Data}}$$

4 Classification Summary

Category	Primary Purpose	Typical Example
Frontend	Change presentation or embed frontend applications	CSS, JavaScript, Custom Element, iFrame
Configuration	Deploy reusable Liferay configuration	OAuth configuration, instance settings
Microservice	Execute business logic outside Liferay	Object Action, Notification, Workflow Action
Batch	Deploy/import structured data entities	Objects, workflows, configuration data

Part I
Frontend Client Extensions

5 Frontend Client Extensions

Frontend Client Extensions provide frontend resources that affect Liferay's presentation layer. They can be used to:

- Add CSS

- Add JavaScript
- Modify theme styling
- Change favicons
- Embed remote applications
- Register web components

Conceptually:

$$\boxed{\text{Frontend CX} \rightarrow \text{Presentation Layer}}$$

Common frontend types include:

- CSS
- JavaScript
- Theme CSS
- Theme Favicon
- Custom Element
- iFrame

6 CSS Client Extension

A CSS Client Extension adds custom styling to Liferay pages.

For example:

```
body {
    font-family: Arial, sans-serif;
}

.custom-card {
    border-radius: 12px;
    padding: 20px;
}
```

Conceptually:

$$\text{Existing Theme} + \text{CSS CX} = \text{Customized UI}$$

Typical uses include:

- Corporate branding
- Typography
- Buttons
- Navigation styling
- Layout enhancements
- Component overrides

A conceptual YAML definition is:

```
my-custom-css:
    name: My Custom CSS
    type: globalCSS
    url: css/main.css
```

Version Awareness

The exact client extension type name and YAML properties should be checked against the Client Extension Reference for the specific Liferay 7.4 update being used.

7 JavaScript Client Extension

A JavaScript Client Extension can inject custom JavaScript behavior.

For example:

```
console.log("Custom Liferay JavaScript loaded.");
```

Possible uses include:

- Analytics integration
- Event listeners
- UI behavior
- External JavaScript libraries
- Custom tracking

Conceptually:

$$\text{Liferay Page} + \text{JavaScript CX} \rightarrow \text{Custom Browser Behavior}$$

8 Theme CSS Client Extension

Theme CSS Client Extensions provide theme-level CSS resources.

They can be used where a more substantial theme-style customization is needed without deploying a traditional Liferay theme.

Conceptually:

```
Liferay Theme
     |
     +--> Main CSS
     |
     +--> Clay CSS
     |
     +--> Theme CSS Client Extension
```

This can reduce the need to maintain a full traditional theme for certain styling requirements.

9 Theme Favicon Client Extension

A Theme Favicon Client Extension changes the favicon displayed by the browser.
For example:

`company-logo.svg`

becomes the site's favicon.
This is useful when branding multiple Liferay environments or sites.

10 Custom Element Client Extension

Custom Element Client Extensions are one of the most powerful frontend extension mechanisms.
They allow externally developed web applications to be registered as widgets in Liferay.
Technologies can include:

- React
- Angular
- Vue
- Vanilla JavaScript
- Web Components

Architecture:

```
+-------------------------+
| Liferay Page            |
|                         |
| +---------------------+ |
| | Custom Element      | |
| |                     | |
| | React / Angular /   | |
| | Vue / JavaScript    | |
| +---------------------+ |
+-------------------------+
```

The application remains decoupled from Liferay's Java runtime.
A conceptual custom element may be:

```
class QuantumDashboard extends HTMLElement {

    connectedCallback() {

        this.innerHTML =
            "<h2>Quantum Dashboard</h2>";
    }
}

customElements.define(
    "quantum-dashboard",
    QuantumDashboard
);
```

Liferay can then expose the custom element as a page widget.

11 iFrame Client Extension

An iFrame Client Extension embeds an externally hosted web application into a Liferay page.

Architecture:



This is useful when:

- An application already exists externally
- Tight frontend integration is unnecessary
- Strong UI isolation is useful
- Migration speed is important

12 Custom Element vs iFrame

	Custom Element	**iFrame**
Integration	Closer	More isolated
UI experience	More native	Embedded
CSS interaction	Possible	Isolated
Existing app reuse	Moderate	Very easy
Complexity	Higher	Lower

Part II
Configuration Client Extensions

13 Configuration Client Extensions

Configuration Client Extensions allow Liferay configuration to be managed and deployed as code.

Conceptually:

Configuration → Version Control → Deployment

Instead of manually repeating configuration across:

```
Development
Testing
UAT
Production
```

configuration can be packaged and deployed consistently.

14 Why Configuration as Code?

Suppose an OAuth application must be configured manually in four environments.

Manual configuration can lead to:

- Human error
- Environment drift
- Missing settings
- Inconsistent permissions

Configuration Client Extensions provide:

Repeatability + Versioning + Automation

15 OAuth Configuration Client Extensions

OAuth-related Client Extensions can support authentication and authorization requirements for other extensions.

For example:

```
Custom Element
      |
      v
OAuth User Agent
      |
      v
Liferay API
```

or:

```
Microservice
      |
      v
OAuth Headless Server
      |
      v
Liferay API
```

OAuth configuration therefore often works together with frontend or microservice Client Extensions.

16 OAuth User Agent

An OAuth User Agent Client Extension is useful for browser/user oriented authorization scenarios.

Conceptually:

User → Frontend CX → OAuth → Liferay API

The authorization context is related to user-triggered activity.

17 OAuth Headless Server

For service-to-service communication, a headless server OAuth configuration can provide machine-oriented access.

Conceptually:

$$\text{External Service} \rightarrow \text{OAuth Token} \rightarrow \text{Liferay Headless API}$$

This is particularly useful for:

- FastAPI services
- Spring Boot services
- Node.js services
- Integration middleware
- AI/ML backends

18 Instance Settings Client Extension

Instance Settings Client Extensions can deploy Liferay instance configuration.

Conceptually:

$$\text{Configuration PID} + \text{Configuration Values} \rightarrow \text{Liferay Instance}$$

This makes configuration repeatable across environments.

Release-Specific Features

Some configuration Client Extension subtypes were introduced in later Liferay 7.4 update releases. Always verify the feature against the exact Liferay DXP quarterly release or Portal GA version.

19 Language Client Extension

In supported newer Liferay releases, Language Client Extensions can add or override language keys.

Conceptually:

```
welcome.message=Welcome to our portal
```

could have translations such as:

```
English -> Welcome
French  -> Bienvenue
Hindi   -> Swagat
```

This can provide language customization without modifying Liferay language bundles directly.

Part III
Microservice Client Extensions

20 Microservice Client Extensions

Microservice Client Extensions integrate external backend logic with Liferay.

The external application runs outside the Liferay JVM.

Liferay ⟷ External Microservice

The microservice can use technologies such as:

- Java / Spring Boot
- Python / FastAPI
- Node.js
- Go
- .NET

21 Microservice Architecture

```
+------------------------+
| Liferay DXP            |
|                        |
| Objects                |
| Workflow               |
| Notifications          |
| Headless APIs          |
+-----------+------------+
            |
            | HTTPS / OAuth 2
            |
            v
+------------------------+
| External Microservice  |
|                        |
| FastAPI / Spring Boot  |
| Node.js / Go / .NET    |
+-----------+------------+
            |
            v
+------------------------+
| DB / AI / External API |
+------------------------+
```

22 Object Action Client Extension

An Object Action Client Extension allows an external service to execute business logic in response to a Liferay Object event/action.

For example:

```
Order Object Created
        |
        v
Object Action
        |
        v
```

```
External Microservice
        |
        v
Fraud Check
```

Another example:

$$\text{Patient Record} \rightarrow \text{Object Action} \rightarrow \text{AI Diagnostic Service}$$

Possible use cases include:

- Validation
- External synchronization
- AI processing
- Payment processing
- Notifications
- ERP integration

23 Workflow Action Client Extension

A Workflow Action Client Extension can connect workflow execution to external logic.
For example:

```
Workflow
   |
   v
Manager Approval
   |
   v
External Compliance Service
   |
   v
Continue Workflow
```

This is useful when workflow processing requires functionality that should not run inside Liferay.

24 Workflow Condition Client Extension

A workflow condition can use external logic to determine the workflow path.
For example:

$$\text{Risk Score} < 50 \Rightarrow \text{Normal Approval}$$

while:

$$\text{Risk Score} \geq 50 \Rightarrow \text{Enhanced Review}$$

The risk score itself could be calculated by an external microservice.

25 Notification Client Extension

A notification-oriented extension can integrate Liferay with an external notification service.

Conceptually:

```
Liferay Event
     |
     v
Notification CX
     |
     v
External Notification Service
     |
     +--> Email
     +--> SMS
     +--> Push
     +--> Enterprise Messaging
```

26 Custom Microservice

A microservice can also implement specialized business logic.

Examples:

- AI inference
- Quantum simulation
- Document processing
- Payment processing
- ERP integration
- CRM integration
- Data transformation

For example:

```
Liferay Portal
      |
      v
FastAPI Service
      |
      v
Qiskit / Qulacs
      |
      v
Quantum Result
      |
      v
Liferay
```

Part IV
Batch Client Extensions

27 Batch Client Extensions

Batch Client Extensions provide data entities to a Liferay instance.

Instead of manually creating resources in every environment:

$$\text{Export} \rightarrow \text{Package} \rightarrow \text{Deploy} \rightarrow \text{Import}$$

can be used.

Typical entities can include:

- Object definitions
- Workflow definitions
- Application data
- Supported configuration/data entities

28 Batch Architecture

```
Development Environment
        |
        | Export
        v
Batch Data
        |
        v
Batch Client Extension
        |
        | Deploy
        v
Target Liferay
        |
        v
Imported Entities
```

29 Batch Engine Data

A built Batch Client Extension can contain files with names such as:

```
*.batch-engine-data.json
```

These are packaged under a batch directory.

Conceptually:

```
batch/
|
+-- object-definitions.batch-engine-data.json
|
+-- workflows.batch-engine-data.json
```

This makes data/configuration transport part of the deployment lifecycle.

30 When to Use Batch Client Extensions

Batch Client Extensions are useful when:

- The same Object definitions are needed across environments
- Workflow definitions must be deployed repeatedly
- Initial application data must be provisioned
- Configuration should be version controlled
- Manual environment setup should be minimized

31 client-extension.yaml

Client Extension projects commonly use:

`client-extension.yaml`

as their primary source configuration.
A single project can conceptually contain multiple extensions:

```
company-css:
    name: Company CSS
    type: globalCSS

company-widget:
    name: Company Widget
    type: customElement

company-service:
    name: Company Service
    type: microservice
```

The actual supported type names and properties depend on the Liferay release.

32 Liferay Workspace Structure

Client Extension projects normally reside under:

```
liferay-workspace/
|
+-- client-extensions/
|   |
|   +-- company-css/
|   |
|   +-- customer-widget/
|   |
|   +-- payment-service/
|   |
|   +-- application-data/
|
+-- modules/
|
+-- themes/
|
+-- gradle/
```

This provides a clear separation between traditional modules and modern Client Extensions.

33 Packaging

Client Extension projects are packaged into deployable archives.

A conceptual packaged structure is:

```
client-extension.zip
|
+-- *.client-extension-config.json
|
+-- static/
|
+-- batch/
|
+-- Dockerfile
|
+-- LCP.json
|
+-- microservice resources
```

The exact contents depend on the Client Extension classification.

For example:

- Frontend extensions use static resources
- Batch extensions contain batch data
- Microservice extensions can contain service resources
- Configuration extensions primarily provide configuration

34 LUFFA

Liferay refers to the deployable Client Extension archive as a Liferay Universal File Format Archive, or LUFFA.

Conceptually:

Client Extension Source → Build → LUFFA (.zip)

The Liferay Workspace tooling normally handles the packaging process.

35 Build and Deployment

A typical build command is:

```
./gradlew build
```

On Windows:

```
gradlew.bat build
```

For a specific Client Extension:

```
gradlew.bat :client-extensions:my-extension:build
```

The deployment process varies according to environment and extension type.
Conceptually:

```
Source
  |
  v
client-extension.yaml
  |
  v
Gradle Build
  |
  v
LUFFA / ZIP
  |
  v
Liferay Environment
```

36 Frontend Packaging

Frontend Client Extensions generally contain static resources.
For example:

```
static/
|
+-- css/
|   +-- main.css
|
+-- js/
|   +-- main.js
|
+-- images/
    +-- logo.svg
```

A frontend container can serve these resources externally to Liferay.

37 Microservice Packaging

A Microservice Client Extension can require its own Dockerfile because the runtime depends on the selected technology.
For example:

```
FROM python:3.12-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

CMD [
    "uvicorn",
    "app.main:app",
```

```
    "--host",
    "0.0.0.0",
    "--port",
    "8080"
]
```

This is fundamentally different from a frontend static-resource container.

38 OAuth 2 and Client Extensions

Client Extensions frequently need authenticated access to Liferay.

The general pattern is:

$$\text{Client Extension} \rightarrow \text{OAuth 2} \rightarrow \text{Liferay API}$$

For frontend/user-oriented scenarios:

```
User
 |
 v
Frontend CX
 |
 v
OAuth User Agent
 |
 v
Liferay API
```

For service-to-service scenarios:

```
Microservice
 |
 v
OAuth Headless Server
 |
 v
Liferay Headless API
```

39 Client Extensions vs OSGi Modules

Characteristic	Client Extension	OSGi Module
Runtime	Generally outside core Liferay runtime	Inside Liferay JVM
Coupling	Loose	Potentially tighter
Languages	Technology dependent / flexible	Primarily Java
Communication	APIs and extension points	Java/OSGi APIs
Scaling	Can be independent	Usually tied to Liferay
Upgrade impact	Generally lower	Can require API adaptation
Internal Liferay APIs	Limited	Available depending on API
Cloud suitability	Strong	Depends on deployment model

40 When Should OSGi Still Be Used?

Client Extensions do not eliminate OSGi.

An OSGi module may still be appropriate when the requirement needs:

- Deep portal internals
- Liferay service overrides
- Internal extension points
- Java service components
- Low-level platform integration
- Functionality unavailable through Client Extension APIs

Therefore:

Use Client Extensions when possible

but:

Use OSGi when deep platform integration is required

41 How to Choose the Right Type

A simple decision model is:

```
What do you need to extend?
        |
        +--> UI / Styling?
        |       |
        |       +--> Frontend CX
        |
        +--> Liferay Configuration?
        |       |
        |       +--> Configuration CX
        |
        +--> External Business Logic?
        |       |
        |       +--> Microservice CX
        |
        +--> Deploy Data / Definitions?
                |
                +--> Batch CX
```

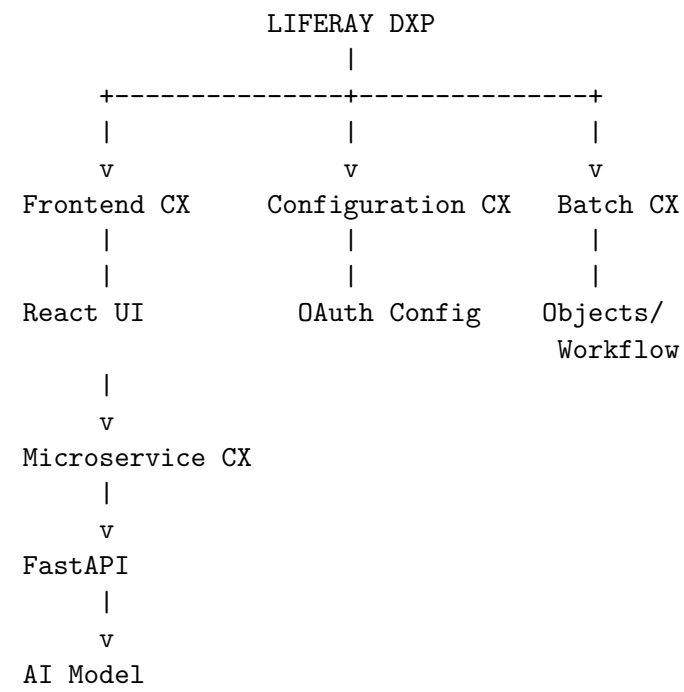
Or mathematically:

$$CX(x) = \begin{cases} Frontend, & x = \text{presentation} \\ Configuration, & x = \text{configuration} \\ Microservice, & x = \text{external logic} \\ Batch, & x = \text{data provisioning} \end{cases}$$

42 Example Composite Application

A real application may use all four categories simultaneously.

Consider an **AI Assessment Portal**.



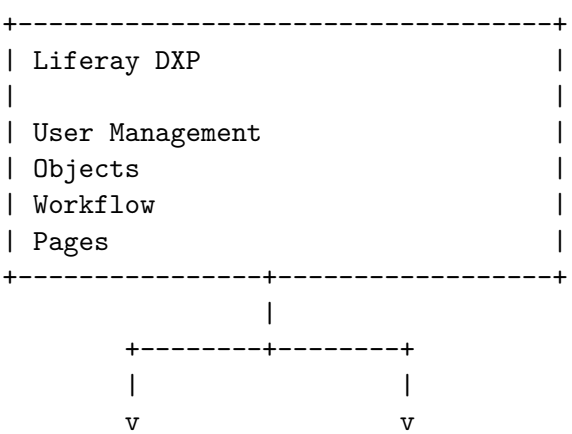
The responsibilities are:

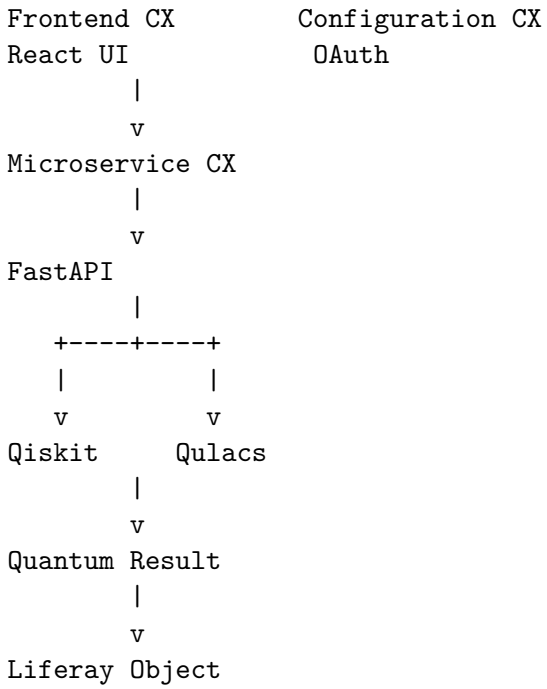
Extension	Responsibility
Frontend	React/JavaScript interface
Configuration	OAuth and Liferay configuration
Microservice	AI/business processing
Batch	Objects/workflows/application data

This demonstrates why Client Extensions should be considered an ecosystem rather than isolated features.

43 Example Quantum Application

For a quantum assessment platform:





A Batch Client Extension could additionally deploy:

- Assessment Object definitions
- Question structures
- Workflow definitions
- Initial configuration data

44 Security Considerations

Client Extensions should follow standard security practices.

Important considerations include:

- HTTPS/TLS
- OAuth 2
- Least privilege
- Secure secret storage
- Input validation
- CORS configuration
- Content Security Policy
- API authorization
- Token expiration
- Audit logging

Never place secrets directly in:

- Browser JavaScript
- Public repositories
- CSS
- Public configuration

45 Advantages

The Client Extension model provides several advantages.

45.1 Loose Coupling

$$\text{Extension} \not\subset \text{Liferay Core}$$

45.2 Technology Freedom

External components can use technologies appropriate to the task.

45.3 Independent Scaling

$$\text{Scale(Liferay)} \neq \text{Scale(Microservice)}$$

45.4 Upgrade Resilience

API-oriented integration can reduce dependencies on Liferay implementation details.

45.5 Cloud Readiness

Externalized applications and configuration-as-code patterns align well with modern cloud architectures.

46 Limitations

Client Extensions are not automatically the best solution for every requirement.

Potential considerations include:

- Network latency
- API limitations
- Authentication complexity
- Distributed debugging
- External service availability
- Version-dependent capabilities
- CORS/security configuration

A deeply internal portal customization may still require OSGi.

47 Troubleshooting

47.1 Client Extension Does Not Appear

Check:

- YAML syntax
- Extension type
- Build output
- Deployment logs
- Liferay version compatibility

47.2 Frontend Resource Not Loaded

Check:

- Resource URL
- Browser console
- Network tab
- MIME type
- CORS
- Content Security Policy

47.3 401 Unauthorized

Check:

- OAuth client configuration
- Access token
- Token expiration
- Client ID
- Client secret

47.4 403 Forbidden

Authentication may have succeeded but the identity does not have the required authorization.

47.5 Microservice Not Reachable

Check:

- Service URL
- DNS
- Firewall
- Container
- Port
- TLS certificate

48 Interview Questions and Answers

What is a Client Extension?

A Client Extension is a loosely coupled mechanism for extending Liferay through supported APIs and extension points without placing the extension's application logic directly inside Liferay's core JVM runtime.

What are the four main classifications?

Frontend, Configuration, Microservice, and Batch Client Extensions.

Which Client Extension is used for React?

A Custom Element Client Extension is a common choice for integrating React applications as Liferay page widgets.

Which Client Extension is used for external backend logic?

Microservice Client Extensions are designed for integrating external business logic and services.

Which Client Extension is used to transport Object definitions?

Batch Client Extensions can package supported data entities such as Object definitions for deployment/import.

Which category handles OAuth configuration?

OAuth-related extensions belong to the Configuration Client Extension category.

What is client-extension.yaml?

It is the main source descriptor used to define the Client Extensions contained in a project.

What is LUFFA?

LUFFA stands for Liferay Universal File Format Archive and represents the deployable archive format used for packaged Client Extensions.

49 Multiple Choice Questions

Question 1

How many major Client Extension classifications does Liferay define?

A. Two

B. Three

C. Four

D. Seven

Answer: C

Question 2

Which category is primarily responsible for presentation?

A. Batch

B. Frontend

C. Microservice

D. Configuration

Answer: B

Question 3

Which Client Extension is suitable for a React application rendered as a Liferay widget?

A. Batch

B. Custom Element

C. Object Action

D. Instance Settings

Answer: B

Question 4

Which Client Extension is most suitable for embedding an existing external website?

A. iFrame

B. Batch

C. Theme Favicon

D. OAuth

Answer: A

Question 5

Which category executes specialized business logic externally?

A. Frontend

B. Microservice

C. Batch

D. Theme

Answer: B

Question 6

Which category is useful for deploying Object definitions and workflow definitions?

A. Batch

B. CSS

C. iFrame

D. JavaScript

Answer: A

Question 7

Which file commonly defines a Client Extension project?

A. service.xml

B. portlet.xml

C. client-extension.yaml

D. web.xml

Answer: C

Question 8

Which category can deploy OAuth configuration?

A. Configuration

B. Batch only

C. CSS

D. Theme

Answer: A

Question 9

Where does an external Microservice Client Extension execute?

A. Inside the Liferay JVM

B. Outside the Liferay JVM

C. Inside Elasticsearch

D. Inside the browser only

Answer: B

Question 10

Which technology can be used for an external microservice?

A. FastAPI

B. Spring Boot

C. Node.js

D. All of the above

Answer: D

Question 11

What is a major advantage of Client Extensions?

A. Stronger dependency on Liferay internals

B. Loose coupling

C. All code must be Java

D. They require portal source modification

Answer: B

Question 12

Which extension can be used to add page-level CSS?

A. CSS Client Extension

B. Batch Extension

C. Workflow Extension

D. OAuth Extension

Answer: A

Question 13

What does LUFFA represent?

A. A Liferay database

B. A packaged Client Extension archive

C. A Java servlet

D. A workflow task

Answer: B

Question 14

Which model is suitable for service-to-service Liferay integration?

A. Microservice + OAuth

B. CSS only

C. Theme Favicon

D. JSP only

Answer: A

Question 15

Client Extensions generally communicate with Liferay through:

A. Supported APIs and extension points

B. Direct database modification

C. JVM memory

D. JSP compilation

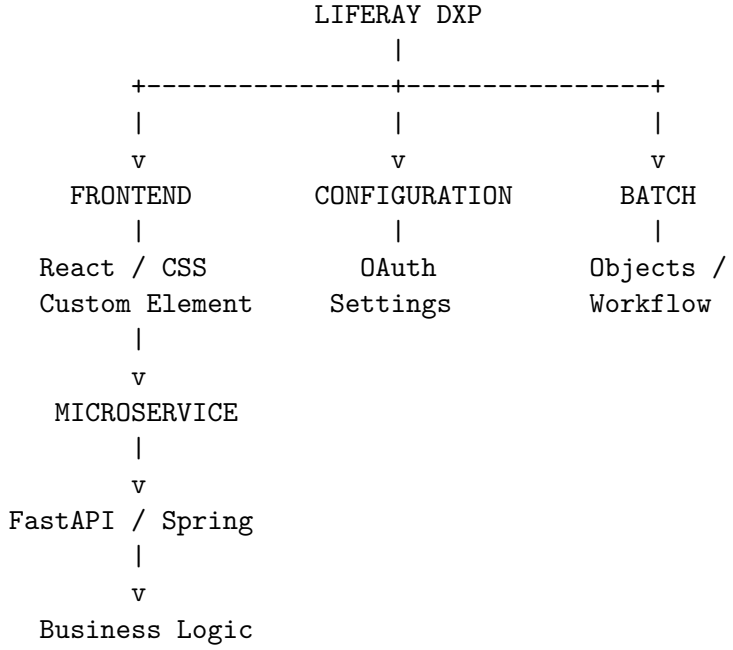
Answer: A

50 Quick Revision Table

Client Extension	Primary Purpose
CSS	Add custom CSS styling
JavaScript	Add browser-side JavaScript behavior
Theme CSS	Provide/override theme styling resources
Theme Favicon	Replace site/theme favicon
Custom Element	Embed modern frontend application as a widget
iFrame	Embed externally hosted application
OAuth User Agent	Support user-oriented OAuth authorization
OAuth Headless Server	Support machine/service-oriented API access
Instance Settings	Deploy Liferay instance configuration
Language	Add/override language keys in supported releases
Object Action	Invoke external logic for Object actions
Workflow Action	Invoke external logic from workflow processing
Workflow Condition	Use external logic for workflow routing
Notification	Connect notification processing to external services
Batch	Deploy/import supported data entities

51 The Big Picture

A modern Liferay application can combine all four categories:



The responsibilities can therefore be separated as:

$$\boxed{\text{Frontend CX} = \text{Presentation}}$$

$$\boxed{\text{Configuration CX} = \text{Platform Configuration}}$$

$$\boxed{\text{Microservice CX} = \text{External Business Logic}}$$

$$\boxed{\text{Batch CX} = \text{Data / Definition Provisioning}}$$

Together:

$$\boxed{\text{Modern Liferay Application} = F + C + M + B}$$

where:

$$F = \text{Frontend}$$

$$C = \text{Configuration}$$

$$M = \text{Microservice}$$

$$B = \text{Batch}$$

52 Key Takeaway

The easiest way to remember Liferay Client Extensions is:

Need	**Choose**	**Think**
Change UI	Frontend	Presentation
Configure Liferay	Configuration	Configuration as Code
Execute external logic	Microservice	Business Logic
Deploy data/definitions	Batch	Provisioning

Therefore:

$$\boxed{\text{UI} \rightarrow \text{Frontend CX}}$$

$$\boxed{\text{Settings} \rightarrow \text{Configuration CX}}$$

$$\boxed{\text{External Logic} \rightarrow \text{Microservice CX}}$$

$$\boxed{\text{Data} \rightarrow \text{Batch CX}}$$

The important architectural shift is from:

$$\text{Customization} = \text{Code Inside Liferay}$$

toward:

$$\boxed{\text{Customization} = \text{Loosely Coupled Extensions} + \text{Stable APIs}}$$

This reduces unnecessary coupling and makes it possible to combine Liferay's portal, Objects, workflow, identity, permissions, and Headless APIs with modern frontend frameworks and independently deployed backend services.

53 Version Considerations

Liferay 7.4 has evolved significantly across GA and quarterly releases.

Therefore, "Liferay 7.4" should not be treated as a single fixed feature set.

Some Client Extension types and capabilities were introduced or enhanced in later 7.4 updates.

Before implementing a particular extension, verify:

- Exact Liferay DXP quarterly release or Portal GA
- Client Extension type availability
- Supported YAML properties
- OAuth capabilities
- Deployment environment
- Feature flags where applicable

54 Official Reference

The official Liferay Client Extensions documentation is available at:

https://learn.liferay.com/w/dxp/development/client-extensions

For exact extension type names and supported properties, also consult the Client Extension Reference corresponding to the target Liferay release.