Liferay 7.4 Client Extensions

Interview & Technical Reference Notes

What are Client Extensions?

Client Extensions are Liferay's mechanism for extending Liferay functionality without requiring the extension code to run inside Liferay's core OSGi runtime.
They provide a cleaner separation between the Liferay platform and custom applications.
A useful mental model is:

$$\boxed{\text{Liferay} \longleftrightarrow \text{Client Extension} \longleftrightarrow \text{External Application / Service}}$$

Client Extensions are particularly important for modern, cloud-oriented and upgrade-friendly Liferay architectures.

Basic Concepts

1. What is a Client Extension in Liferay 7.4?

Answer: A Client Extension is a mechanism for extending or integrating with Liferay while keeping much of the custom implementation outside the traditional Liferay OSGi runtime.

Instead of modifying Liferay internals or deploying all customization logic as OSGi bundles, Client Extensions allow developers to connect external applications, frontend resources, services, OAuth clients, and other capabilities to Liferay through supported extension points.

Conceptually:

$$\boxed{\text{Extension Without Tight Platform Coupling}}$$

This makes Client Extensions attractive for:

- External microservices
- React applications
- JavaScript/CSS customization
- Remote applications
- OAuth integrations
- External notification systems
- Custom element applications
- Headless architectures

2. Why were Client Extensions introduced?

Answer: Traditional Liferay customization frequently relies on OSGi modules deployed directly into the Liferay runtime.

This provides significant power but also creates tighter coupling:

$$\text{Custom Code} \rightarrow \text{Liferay APIs} \rightarrow \text{Liferay Runtime}$$

Client Extensions enable a more loosely coupled model:

$$\text{Liferay} \longleftrightarrow \text{HTTP / OAuth / JS / CSS} \longleftrightarrow \text{External Extension}$$

The main architectural goals include:

- Better separation of concerns
- Easier platform upgrades
- Reduced dependence on Liferay internals
- Cloud-friendly deployment
- Independent development lifecycles
- Independent technology stacks

3. What is the main difference between Client Extensions and OSGi modules?

Answer:

An OSGi module executes **inside** the Liferay application runtime.

A Client Extension generally integrates through supported external extension mechanisms rather than embedding all application logic inside that runtime.

$$\boxed{\text{OSGi} \Rightarrow \text{Inside Liferay}}$$

$$\boxed{\text{Client Extension} \Rightarrow \text{Loosely Coupled / External}}$$

4. What are common Client Extension types?

Answer: Liferay 7.4 supports several Client Extension categories, depending on the product update and deployment environment.

Important categories include:

- Custom Element
- Global CSS
- Global JavaScript
- OAuth 2 Application
- Object Action
- Object Validation
- Notification Type
- Theme CSS
- Theme Favicon
- Theme JavaScript
- Theme Spritemap

Some Client Extension types are especially relevant to Liferay's cloud and SaaS-oriented architecture.

Client Extension Project Structure

5. What does a Client Extension project look like?

Answer: A Client Extension project commonly contains configuration describing the extension together with the resources or source code required by that extension.

A simplified structure might look like:

```
client-extensions/
|
+-- employee-dashboard/
    |
    +-- client-extension.yaml
    +-- package.json
    +-- src/
    |   +-- index.js
    |
    +-- assets/
```

The most important configuration file is typically:

```
client-extension.yaml
```

It describes how Liferay should recognize and configure the extension.

6. What is client-extension.yaml?

Answer: The `client-extension.yaml` file is the primary descriptor used to define Client Extensions.

A simplified conceptual example is:

```
employee-dashboard:
    type: customElement
    name: Employee Dashboard
    htmlElementName: employee-dashboard
    urls:
        - index.js
```

The exact supported properties depend on the Client Extension type and the installed Liferay version.

Therefore:

`client-extension.yaml` = Client Extension Descriptor

Intermediate Concepts

7. What is a Custom Element Client Extension?

Answer: A Custom Element Client Extension allows a frontend application to be exposed inside Liferay as a browser custom element.

For example:

```
<employee-dashboard></employee-dashboard>
```

The implementation could be created using technologies such as:

- React
- JavaScript
- TypeScript
- Web Components

- Other frontend frameworks

Conceptually:

Liferay Page → Custom Element → Frontend Application

This allows the frontend application to evolve with less coupling to Liferay's Java/OSGi implementation.

8. What is a Global CSS Client Extension?

Answer: A Global CSS Client Extension loads custom CSS resources into configured Liferay contexts.

It can be useful for:

- Branding
- Corporate styling
- Common visual customization
- Portal-wide CSS adjustments

For example:

```
.portal-header {
    font-weight: 600;
}

.company-banner {
    border-radius: 8px;
}
```

Instead of modifying Liferay's internal theme source directly, the custom CSS can be managed as an extension.

9. What is a Global JavaScript Client Extension?

Answer: A Global JavaScript Client Extension allows custom JavaScript resources to be introduced into selected Liferay contexts.

Typical use cases include:

- Analytics integration
- Client-side event handling
- Third-party scripts
- Common browser-side behavior

However, global JavaScript should be used carefully because code executed globally can affect multiple pages.

10. What is an OAuth 2 Client Extension?

Answer: OAuth 2 Client Extensions are particularly important when an external application needs secure access to Liferay APIs.

Instead of embedding credentials directly into application code, OAuth 2 can be used to establish controlled API access.

Conceptually:

External Application

↓

OAuth 2 Authentication

↓

Liferay Headless API

For example, a Spring Boot microservice may authenticate and then call:

```
GET /o/headless-delivery/v1.0/sites/{siteId}/blog-postings
```

or a custom Object API such as:

```
GET /o/c/employees
```

Client Extensions and Headless APIs

11. How do Client Extensions fit into a Headless Liferay architecture?

Answer: Client Extensions work naturally with Liferay's Headless APIs.

A modern architecture might look like:

React Client Extension

↓

Liferay Headless REST API

↓

Liferay Objects / Content / Users

The frontend does not need direct access to Liferay's internal Java services.

Instead, it communicates through supported APIs.

12. Can a Client Extension call Liferay Objects?

Answer: Yes.

This is an important architectural combination.

Suppose we have an Employee Object.

The generated API could expose an endpoint similar to:

```
GET /o/c/employees
```

A React-based Client Extension could retrieve the entries:

```
fetch('/o/c/employees')
    .then(response => response.json())
    .then(data => {
        console.log(data.items);
    });
```

Conceptually:

React Client Extension → /o/c/employees → Employee Object

This can provide a low-code backend together with a custom frontend.

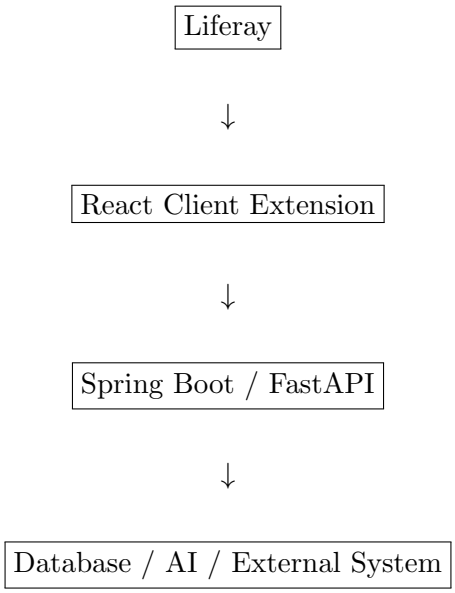
Advanced Concepts

13. Can a Client Extension communicate with an external microservice?

Answer: Yes.

One of the major advantages of the Client Extension architecture is the ability to integrate Liferay with independently deployed services.

For example:



The backend does not necessarily need to be implemented using Java.
Possible technologies include:

- Spring Boot
- FastAPI
- Node.js
- .NET
- Go
- Serverless functions

14. How is security handled?

Answer: Security depends on the Client Extension and integration architecture.

Important mechanisms can include:

- OAuth 2
- Liferay permissions
- API authorization
- HTTPS
- CORS configuration
- Secure credential management
- External service authentication

A common machine-to-machine pattern is:

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

The key principle is:

$$\boxed{\text{Never expose privileged secrets in browser JavaScript}}$$

Sensitive credentials should be handled by trusted backend services.

15. Can Client Extensions use Liferay permissions?

Answer: Yes, when the extension accesses Liferay resources through Liferay's APIs, the API operation remains subject to the applicable authentication and permission checks.

For example:

$$\text{User} \rightarrow \text{Client Extension} \rightarrow \text{REST API} \rightarrow \text{Permission Check} \rightarrow \text{Resource}$$

A frontend Client Extension should therefore not be considered a security boundary.

Hiding a button does not remove the user's backend permission.

16. What are Object Action Client Extensions?

Answer: Client Extensions can be combined with Liferay Objects to execute external business logic.

Consider:

$$\begin{array}{c}\text{Purchase Request Created}\\ \downarrow\\ \text{Object Action}\\ \downarrow\\ \text{External Microservice}\\ \downarrow\\ \text{ERP / SAP / Email / Business System}\end{array}$$

This approach helps keep sophisticated integration logic outside the Liferay runtime.

17. What is an Object Validation Client Extension?

Answer: An Object Validation Client Extension can support externally implemented validation behavior for Object entries where the corresponding Client Extension mechanism is available.

For example, consider:

$$\text{Purchase Amount} = 2,000,000$$

The validation service could check:

- Department budget
- Vendor status
- Spending policy
- External finance rules

Conceptually:

$$\boxed{\text{Object Entry}} \rightarrow \boxed{\text{Validation Extension}} \rightarrow \boxed{\text{External Logic}}$$

This is valuable when validation is too sophisticated to represent using simple configuration.

Client Extensions vs OSGi Modules

18. When should Client Extensions be used instead of OSGi?

Feature	Client Extension	OSGi Module
Execution	Often external/browser-based	Inside Liferay JVM
Coupling	Generally lower	Can be tightly coupled to Liferay Java APIs
Technology	React, JS, Spring Boot, FastAPI, etc.	Primarily Java/OSGi
Deployment	Extension-oriented deployment model	OSGi bundle deployment
Internal Services	Normally uses APIs/integration contracts	Can directly use Liferay Java services
Upgrade Isolation	Generally stronger	Depends on APIs used by custom code
Microservices	Natural fit	Not its primary purpose
Deep Platform Logic	Limited compared with OSGi	Strong

A useful rule is:

$$\boxed{\text{Remote / UI / Integration Requirement} \Rightarrow \text{Consider Client Extension}}$$

whereas:

$$\boxed{\text{Deep Liferay JVM Customization} \Rightarrow \text{Consider OSGi}}$$

Client Extensions vs Objects

19. What is the difference between Liferay Objects and Client Extensions?

Answer: They solve different problems and are frequently used together.

	Objects	Client Extensions
Primary Purpose	Business data modeling	Platform extension/integration
Database Entity	Yes	Not inherently
CRUD	Generated	Implemented or consumed as required
REST API	Generated for custom objects	Can consume or integrate with APIs
Frontend	Generated/admin interfaces	Custom UI possible
External Logic	Can trigger/integrate with it	Strong integration mechanism

Therefore, a powerful pattern is:

$$\boxed{\text{Objects} + \text{Client Extensions} + \text{Headless APIs}}$$

Real-World Example: Employee Portal

20. How would you build an Employee Portal using Client Extensions?

Suppose the requirement is:

> Create an Employee Portal where employees can view their profile, submit leave requests and see their leave history.

Step 1: Create Liferay Objects

Create:

```
Employee
LeaveRequest
```

Employee fields:

```
employeeId
firstName
lastName
email
department
designation
```

Leave Request fields:

```
startDate
endDate
leaveType
reason
status
```

Step 2: Create the Relationship

One employee may have many leave requests:

$$\boxed{\text{Employee} \xrightarrow{1:N} \text{LeaveRequest}}$$

Step 3: Use Generated APIs

The published Objects expose APIs conceptually similar to:

```
GET  /o/c/employees
GET  /o/c/leaverequests
POST /o/c/leaverequests
```

Step 4: Create a React Client Extension

Create:

```
employee-dashboard
```

The UI might contain:

```
Employee Dashboard

+----------------------------------+
| Welcome, Employee                |
+----------------------------------+
| Profile                          |
| Leave Balance                    |
| Apply for Leave                  |
| Leave History                    |
+----------------------------------+
```

Step 5: Connect React to Objects

Conceptually:

React → Object REST API → Liferay Objects

Example:

```
fetch('/o/c/leaverequests')
    .then(response => response.json())
    .then(data => {
        console.log(data.items);
    });
```

Step 6: Add Workflow

A Leave Request can follow:

Submitted → Manager Review → Approved / Rejected

Step 7: Add External Integration

Suppose approved leave must also reach an HR system.

The architecture could become:

Leave Approved

↓

Object Action

↓

Client Extension / Webhook Integration

↓

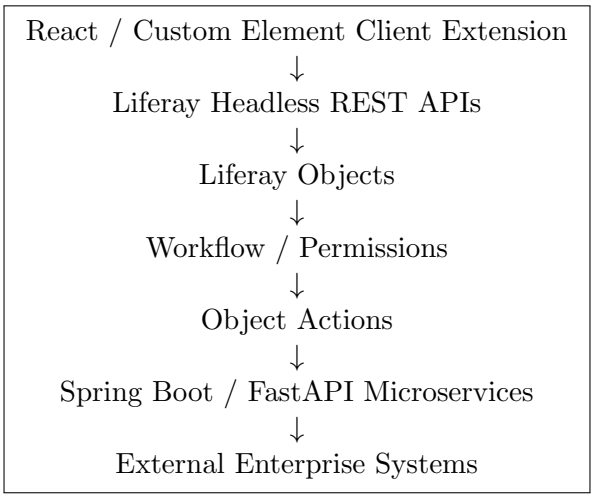
HR Microservice

↓

External HR System

Modern Liferay Architecture

A modern Liferay application can therefore combine:



This architecture separates:

- Presentation
- Business data
- Workflow
- Security
- Integration
- External business logic

Common Interview Questions

21. Does a Client Extension run inside the Liferay JVM?

Answer: Not necessarily. A major purpose of Client Extensions is to allow functionality to be implemented without placing all custom code inside the Liferay JVM.

22. Can Client Extensions replace all OSGi modules?

Answer: No.

Client Extensions are excellent for external integrations, frontend customizations and loosely coupled architectures.

OSGi remains appropriate when deep access to Liferay's Java services, runtime behavior, or internal extension points is required.

23. Can Client Extensions use React?

Answer: Yes. Custom Element Client Extensions are commonly used to integrate modern frontend applications with Liferay.

24. Can Client Extensions call external APIs?

Answer: Yes, provided that authentication, authorization, networking and CORS requirements are correctly configured.

25. Can a Spring Boot application act as part of a Client Extension architecture?

Answer: Yes.

A common pattern is:

$$\boxed{\text{Liferay}} \longleftrightarrow \boxed{\text{Spring Boot Service}}$$

with communication through HTTP APIs and OAuth 2.

26. Can FastAPI be used?

Answer: Yes.

The external service does not have to use Java.

For example:

$$\boxed{\text{Liferay Client Extension}} \rightarrow \boxed{\text{FastAPI}} \rightarrow \boxed{\text{AI/ML Service}}$$

This architecture is particularly useful for AI, machine learning, scientific computing and specialized Python workloads.

Architectural Decision Guide

Which Liferay Technology Should I Use?

Use Liferay Objects when:

$$\text{Requirement} \approx \text{Business Entity} + \text{CRUD} + \text{Workflow}$$

Examples:

- Employee directory
- Purchase requests
- Leave requests
- Customer requests

Use Client Extensions when:

$$\text{Requirement} \approx \text{Custom UI} + \text{External Integration} + \text{Loose Coupling}$$

Examples:

- React dashboard
- External microservice
- Global CSS/JavaScript
- OAuth integration

Use OSGi when:

$$\text{Requirement} \approx \text{Deep Liferay Runtime Customization}$$

Examples:

- Service wrappers
- Model listeners
- Custom Java services
- Internal platform extension points

Interview Takeaway

Key Point

Client Extensions represent an important architectural direction in modern Liferay development. Instead of putting every customization directly inside Liferay's JVM, developers can create independently managed frontend applications and backend services and integrate them with Liferay through supported extension points, APIs, and OAuth.
A particularly powerful modern pattern is:

$$\boxed{\text{Liferay Objects} + \text{Client Extensions} + \text{Headless APIs} + \text{External Microservices}}$$

Objects provide the **business data model**, Client Extensions provide the **extension and presentation layer**, Headless APIs provide the **communication layer**, and external services can provide **specialized business logic**.
OSGi remains important when an application genuinely requires deep integration with the Liferay Java runtime.