1️⃣ XPgListComponent

The `xpglist` component in the XUI framework is responsible for displaying lists of entities in the UI. It dynamically loads schemas, defines properties, and fetches data based on configurations.

Code Structure:

```
loadComponent() {
    this.component = XPgListComponent;
    this.injector = Injector.create({
      providers: [
        {
          provide: XPgListParamProvider,
          useValue: new XPgListParam({
            entityName: 'template',
            detailsComp: TemplateDetComponent,
            onInit: (instance: XPgListComponent) => {},
            sidebarFullSize: true,
          } as XpgListPageConfig),
        },
      ],
      parent: this.inj,
    });
  }
```

- **Component Assignment**: The `this.component` assignment sets the component type to `XPgListComponent`, indicating that this is the component to be rendered for displaying a list of items.

  Injector Creation: An `Injector` instance is created to provide dependencies for the `XPgListComponent`.

- **Providers**: A provider is defined to supply an instance of `XPgListParamProvider`.
- **XPgListParamProvider**: An abstract class used as a base for providing list parameters in the component. It holds the `xpgListPageConfig` object, which contains configuration and data for list management.
- **XPgListParam**: A concrete class extending `XPgListParamProvider`. It provides a specific implementation of the `XPgListParamProvider` service, initialized with `xpgListPageConfig`. This class is used to pass list configuration to the component.
- **xpgListPageConfig**: An interface defining the structure and properties required for list management,

Properties of `xpgListPageConfig`:

Property	Type	Function
`entityName`	string	Specifies the entity the component will interact with. It is used to determine the appropriate backend API endpoints for data fetching.
`columns`	Property[]	Defines an array of `Property` objects that specify additional configurations and customizations for each property of the entity to be displayed as a column. This includes specifying custom templates (`cellTemplate`), custom names (`name`), and other configurations like hiding columns (`hidden`).

`onInit`	(instance?: any) ⇒ void	A callback function that is executed when the component is initialized. It provides access to the component instance, allowing for custom initialization logic
`onListChange`	() ⇒ void	A callback function that is triggered when there is a change in the list data. It allows for custom logic to be executed in response to modifications in the list, such as updating data, filtering items, or adjusting UI elements based on the new state of the list.
`readComp`	Type<any>;	Specifies a component used for displaying the read-only view of an entity. This component allows users to view details without modifying or interacting with the data.
`detailsComp`	Type<any>;	Specifies a component used for displaying the editable or action-oriented view of an entity. This component allows users to modify, edit, or perform actions related to the entity.
`sidebarFullSize`	boolean	Determines if the sidebar should take up the full width of the screen (`true`) or a smaller portion of it (`false`). Enhances layout flexibility and adjusts the UI based on design requirements.
`parentEntity`	string	Specifies the entity name of the parent item.
`parentId`	string	Holds the identifier of the parent item.
`parentProp`	string	Specifies a particular property of the parent entity.
`parentPageMode`	PageMode	Specifies the mode of the parent page, which can be 'create', 'update', or 'read'.
`parentItem`	any	Holds the entire parent item object.
`filterTpl`	TemplateRef<any>	Specifies the template to use for custom filtering in different parts of the component.
`filterBeforeTpl`	TemplateRef<any>	Defines a custom template for filtering before the main filter template is applied.

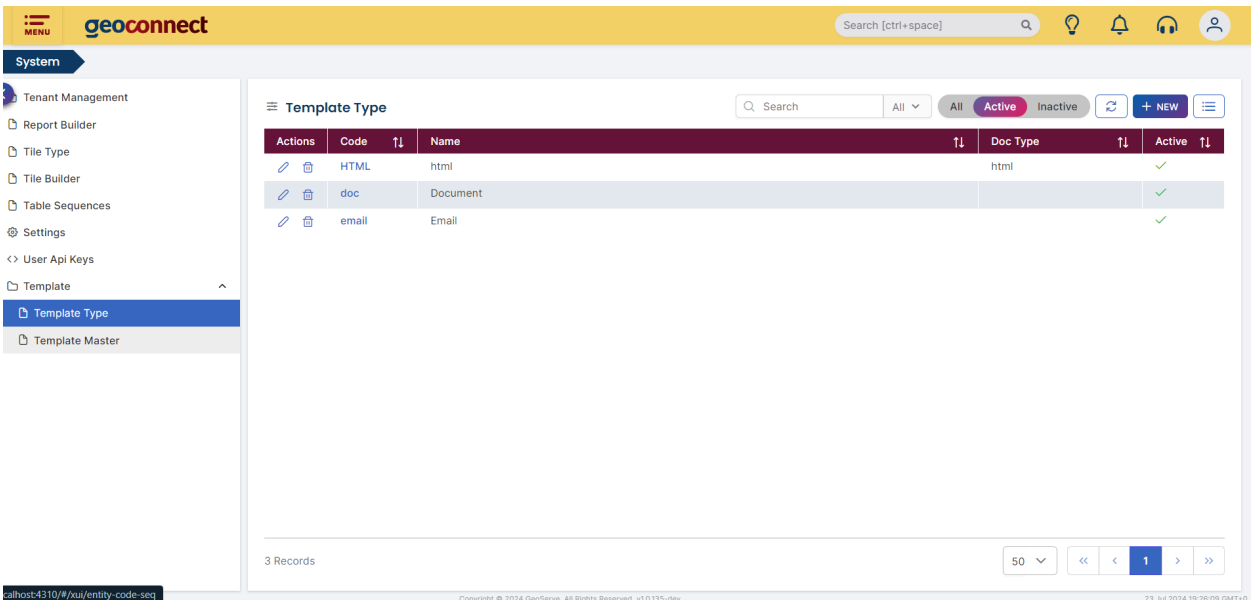
`afterActionTemplate`	TemplateRef<any>;	- -
`headerTpl`	TemplateRef<any>;	A template reference that defines the custom header content for the component. This template can include any HTML or Angular components and is typically used to customize the appearance and functionality of the header section of a list, table, or page.
`replaceTitleTpl`	TemplateRef<any>;	
`staticFilter`	string	A static filter applied to the list query to filter the items based on a specific condition.
`update`	(row: any) ⇒ void;	A function that updates a specified row's details in a list.
`read`	(row: any) ⇒ void;	A function that reads the details of a specified row.
`onDataRequest`	(query: any) ⇒ any;	- -
`replaceListTpl`	TemplateRef<any>;	
`requestMethod`	string	- -
`disablePaginator`	boolean	A flag to enable or disable the pagination control for the list. If `true`, pagination is disabled.
`enableCheckBox`	boolean	A flag to enable or disable checkboxes in the list.
`checkboxTpl`	TemplateRef<any>	A template reference for the checkbox column, defining the structure and behavior of the checkboxes.
`rowClass`	{ [x: string]: string }	- -
`hiddenColumns`	string[]	An array of strings specifying the columns to be hidden in the UI. These columns will not be displayed in the table but can still be part of the data set and used programmatically. In the project it uses `defaultHiddenColumns.filter` to specify columns that should not be displayed.

Output

Here is a description for the `XPgListComponent` usage based on the screenshot and provided details:

Template Management Overview

In the example shown, the `XPgListComponent` is used in the `template.component.ts` to manage and display various template types. This page is part of the Template Management system and includes sections for Template Types and Template Master.



- **Template Types**: This section displays a list of different template types such as HTML, Document, and Email. Each template type is listed with attributes like code, name, and document type.
- **Actions**: For each template type, users can perform actions such as editing or deleting the template type. These actions are enabled by the `XPgListComponent` to facilitate interaction with the template data.

- **UI Elements**: The page includes UI elements like tables, action buttons (edit, delete), and filters to help users manage template types efficiently. The table displays the data in a structured format, allowing users to view and interact with the template types easily.