Backend interface

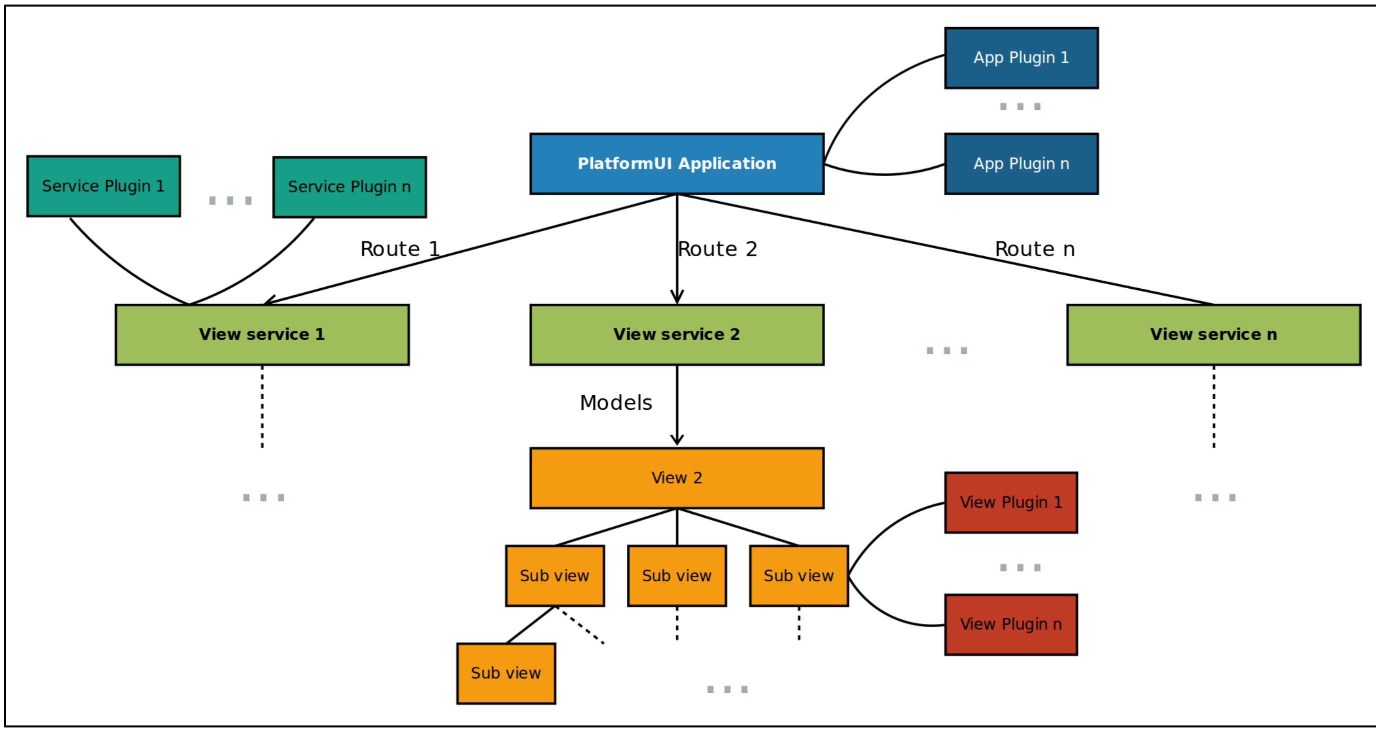
The backend interface is produced by the PlatformUI Bundle which provides a JavaScript Single Page Application based on the YUI App Framework. This application is accessible in your browser at **http://[uri_of_platform]/ez**.

Technical architecture

The PlatformUI application code is divided into different types of components:

- **Application:** this is the top level component, the PlatformUI application is an instance of it. It is responsible for authenticating the user and for handling the routing.
- **Models:** models are the main objects handled by the application, they represent our main domain objects (Content, Location, Content Type, etc.)
- **View services:** view services act between the Application and the Views. They are configured on the routes and the main responsibility of a view service is to provide the model (or other data) to the views and to perform the operations requested by the user (removing a Content item, copying, etc.)
- **Views:** views generate the user interface and handle the user interaction (clicking, form submitting, etc.). A view can have several sub-views which can have further sub-views themselves.
- **Plugins:** plugins can enhance the application, the view services or the views for instance to provide additional features or to tweak the behavior of the plugged component.

The following chart depicts the interaction between those components:



Views: main view, sub-view, side view

The views represent a large part of the application and each of them can be used in three different contexts:

1. As the main view
2. As a sub-view of another (sub-)view
3. As a side view

Main view

A view used as a main view is configured at the route level to be displayed when the user navigates to that route.

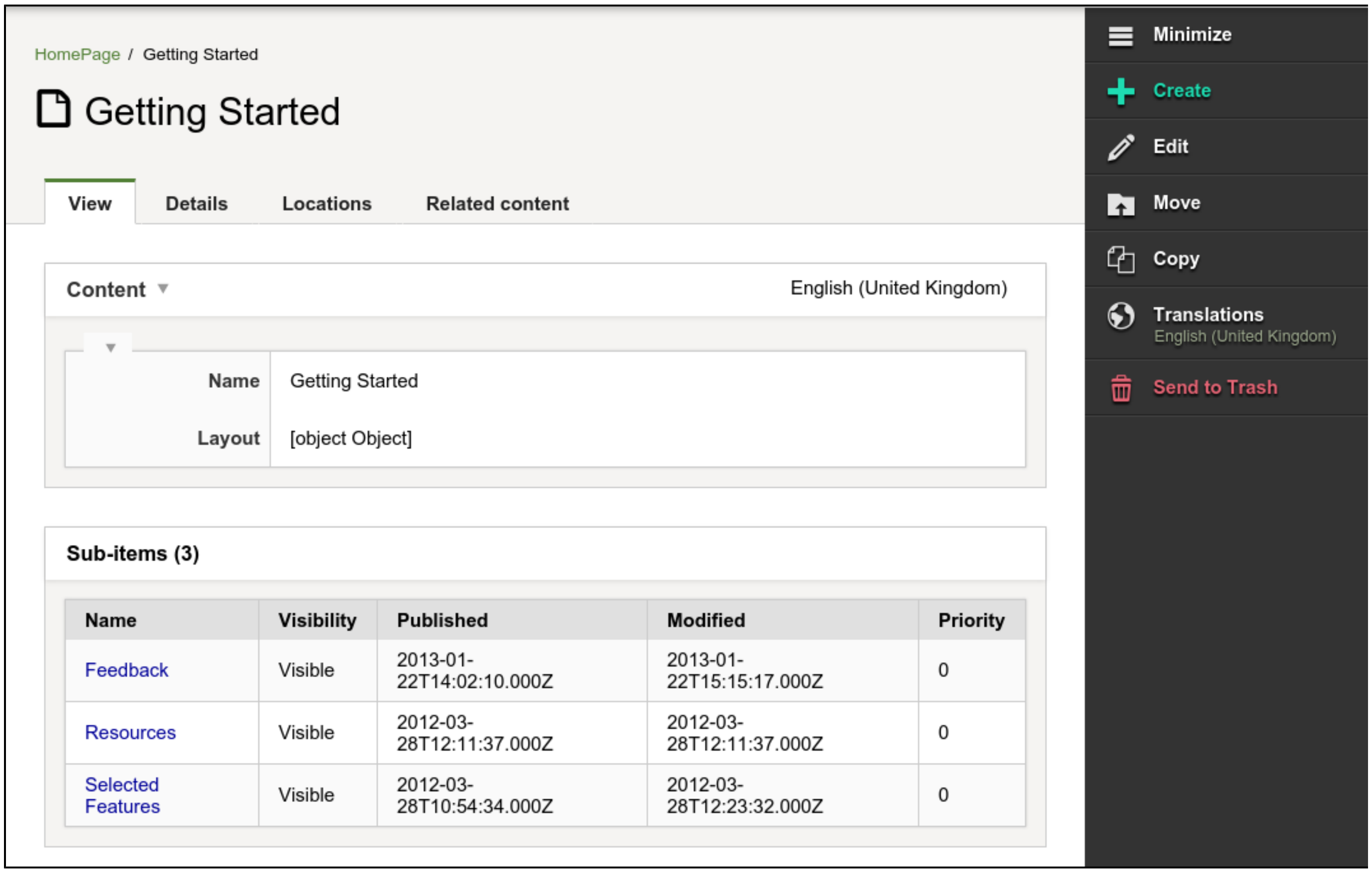
For instance, when reaching `/ez`, the user is redirected to the `loginForm` route (`/ez#/login`) and this route is configured in the following way in the application component:

```
{
      name: "loginForm",
      path: "/login",
      service: Y.eZ.LoginFormViewService,
      sideViews: {'navigationHub': false, 'discoveryBar': false},
      view: 'loginFormView',
      callbacks: ['open', 'handleSideViews', 'handleMainView']
}
```

Among others things, this means the view `loginFormView` will be used as the main view when this route is matched. `loginFormView` is actually the identifier of the view metadata registered in the `view` property of the Application.

Sub-view

To avoid having huge main views doing too many things in the application, the views are divided into smaller parts called sub-views.



Content ▾ English (United Kingdom)

Name	Getting Started
Layout	[object Object]

Sub-items (3)

Name	Visibility	Published	Modified	Priority
Feedback	Visible	2013-01-22T14:02:10.000Z	2013-01-22T15:15:17.000Z	0
Resources	Visible	2012-03-28T12:11:37.000Z	2012-03-28T12:11:37.000Z	0
Selected Features	Visible	2012-03-28T10:54:34.000Z	2012-03-28T12:23:32.000Z	0

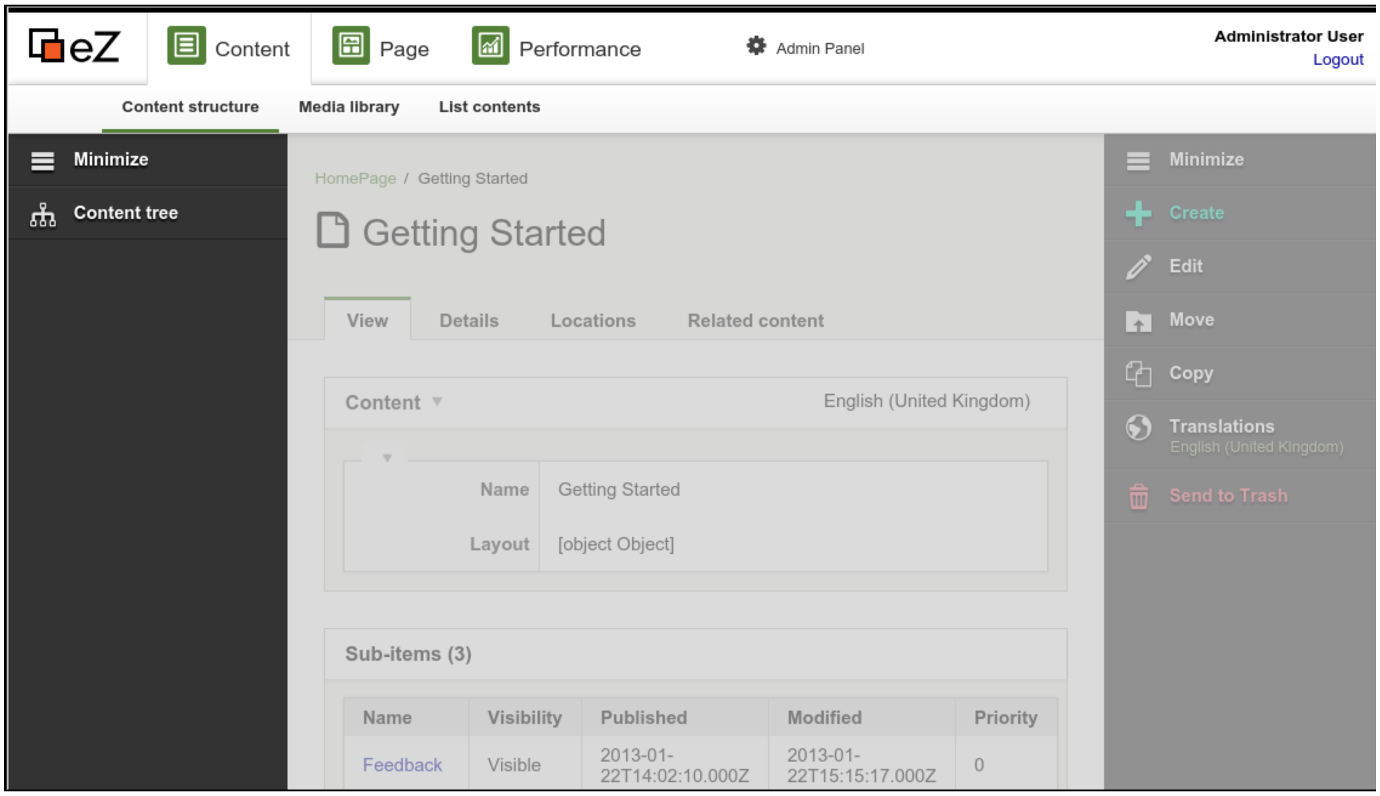
For instance, the view used to display a Location is divided into several views at several levels, it contains:

- An action bar view for the right toolbar, which contains:
    - a view for the Minimize button
    - a view for the Create button which contains:
        - a view to list and select a Content Type
    - a view for the Edit button
    - ...
- A Location View tab view which contains:
    - the Raw Content View to display the fields which contains:
        - A view for each fields
        - ...
- A Location Details tab view
- ...
- A sub-item list view

Side view

A view can also be used as a side view. As its name suggests a side view can represent anything that is not part of the main view.

For instance, when displaying a Location, the top menu (the Navigation hub) or the left toolbar (the Discovery Bar) are side views.



Name	Getting Started
Layout	[object Object]

Name	Visibility	Published	Modified	Priority
Feedback	Visible	2013-01-22T14:02:10.000Z	2013-01-22T15:15:17.000Z	0

The side views are also used for various widgets providing a service used several times in the application, such as the Universal Discovery Widget.

View services

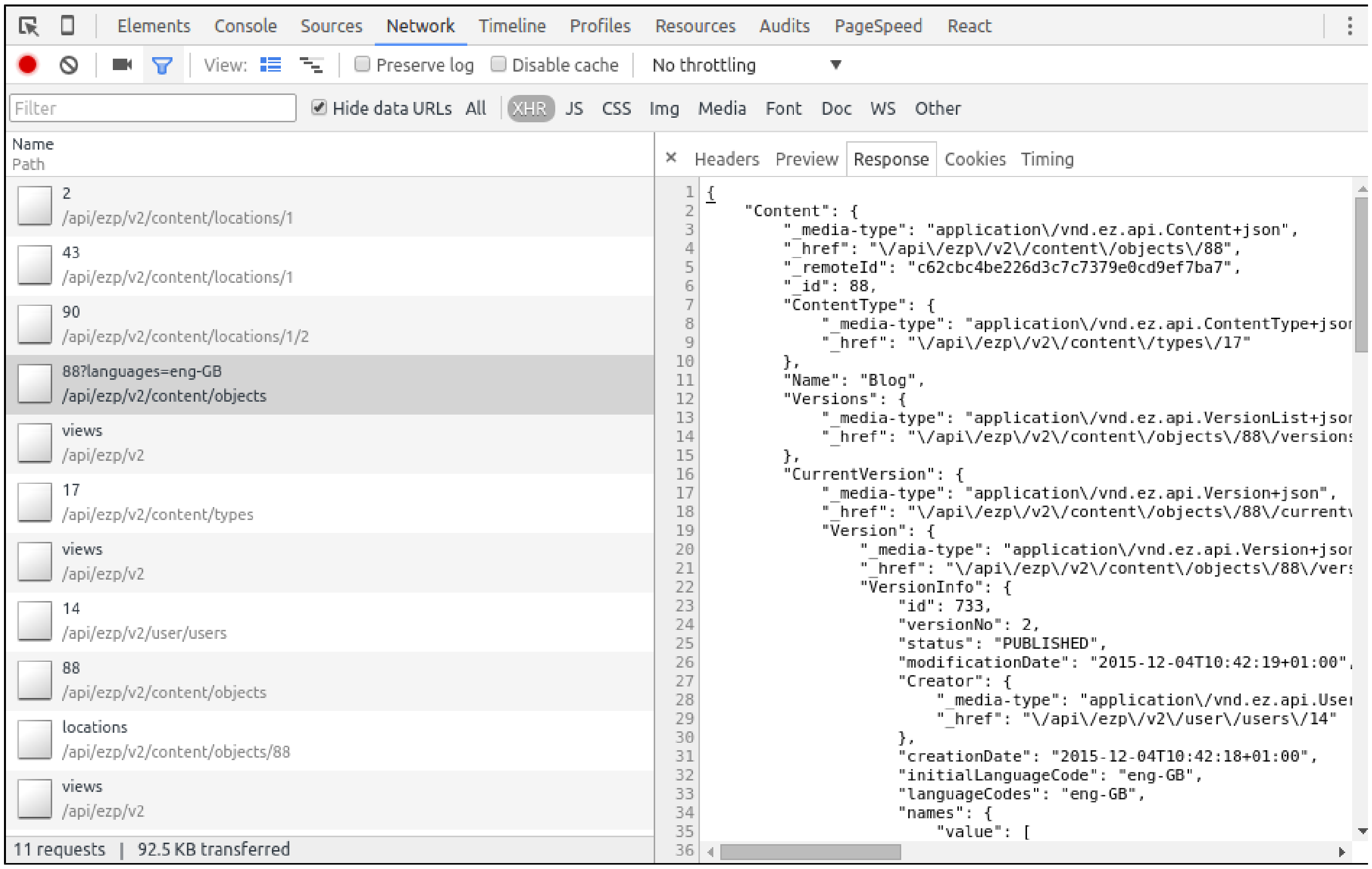
The view services act between the Application and the Views for both the main views and the side views. They are responsible for providing the required data needed by a main view or a side view to be rendered. A view service will also receive the events triggered by the view to react or provide the additional data. For that, the view services receive an instance of the JavaScript REST Client.

How are pages generated?

Depending on the part of the PlatformUI Application you are using, the page may be generated in two different ways. From an end-user perspective, this is almost transparent but as a developer it is important to understand how the page is generated to be able to extend it.

Browser side rendering

The pages in the content part (as opposed to admin related pages) are fully rendered in the browser. For instance, when displaying a Location in PlatformUI, the corresponding view service loads the Location model and the *related* models (Content, Content Type, etc.) with the eZ Platform REST API (through the JavaScript REST Client) and gives them the LocationView to be displayed directly by this view and/or by its sub-views. If you open the browser developer tools in the network panel, you can see the REST requests needed to build the page and they only contain a JSON structure.



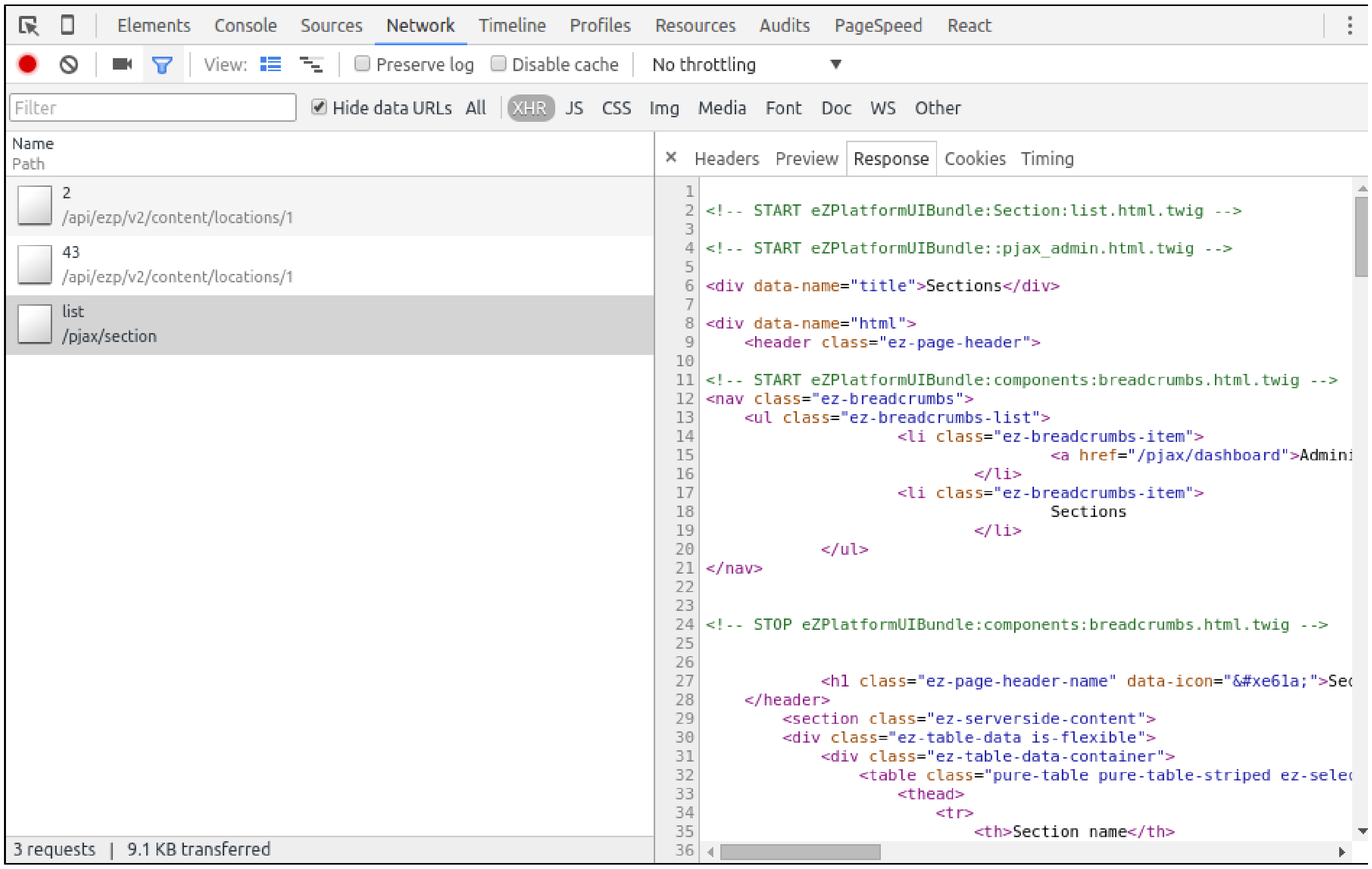
Name / Path
2 /api/ezp/v2/content/locations/1
43 /api/ezp/v2/content/locations/1
90 /api/ezp/v2/content/locations/1/2
88?languages=eng-GB /api/ezp/v2/content/objects
views /api/ezp/v2
17 /api/ezp/v2/content/types
views /api/ezp/v2
14 /api/ezp/v2/user/users
88 /api/ezp/v2/content/objects
locations /api/ezp/v2/content/objects/88
views /api/ezp/v2

11 requests | 92.5 KB transferred

```
{
    "Content": {
        "_media-type": "application\/vnd.ez.api.Content+json",
        "_href": "\/api\/ezp\/v2\/content\/objects\/88",
        "_remoteId": "c62cbc4be226d3c7c7379e0cd9ef7ba7",
        "_id": 88,
        "ContentType": {
            "_media-type": "application\/vnd.ez.api.ContentType+jso
            "_href": "\/api\/ezp\/v2\/content\/types\/17"
        },
        "Name": "Blog",
        "Versions": {
            "_media-type": "application\/vnd.ez.api.VersionList+jso
            "_href": "\/api\/ezp\/v2\/content\/objects\/88\/version
        },
        "CurrentVersion": {
            "_media-type": "application\/vnd.ez.api.Version+json",
            "_href": "\/api\/ezp\/v2\/content\/objects\/88\/current
            "Version": {
                "_media-type": "application\/vnd.ez.api.Version+jso
                "_href": "\/api\/ezp\/v2\/content\/objects\/88\/ver
                "VersionInfo": {
                    "id": 733,
                    "versionNo": 2,
                    "status": "PUBLISHED",
                    "modificationDate": "2015-12-04T10:42:19+01:00",
                    "Creator": {
                        "_media-type": "application\/vnd.ez.api.Use
                        "_href": "\/api\/ezp\/v2\/user\/users\/14"
                    },
                    "creationDate": "2015-12-04T10:42:18+01:00",
                    "initialLanguageCode": "eng-GB",
                    "languageCodes": "eng-GB",
                    "names": {
                        "value": [
```

Server side rendering

The pages in the admin are build in a more traditional way as they are partly rendered server side. For those pages, the view service fetches one (or several) HTML fragment(s) from the server. This HTML fragment follows a very simple structure and can be generated by any means on the server and of course, in PlatformUI this is done in a quite standard Symfony controller. By opening the browser developer tools in the network panel you can see the requests needed to build the section list page.



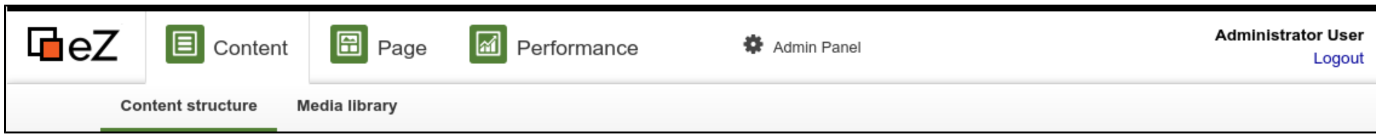
```
2
/api/ezp/v2/content/locations/1
43
/api/ezp/v2/content/locations/1
list
/pjax/section
3 requests | 9.1 KB transferred
```

```

<!-- START eZPlatformUIBundle:Section:list.html.twig -->

<!-- START eZPlatformUIBundle::pjax_admin.html.twig -->

<div data-name="title">Sections</div>

<div data-name="html">
    <header class="ez-page-header">

<!-- START eZPlatformUIBundle:components:breadcrumbs.html.twig -->
<nav class="ez-breadcrumbs">
    <ul class="ez-breadcrumbs-list">
                <li class="ez-breadcrumbs-item">
                                <a href="/pjax/dashboard">Admini
                        </li>
                <li class="ez-breadcrumbs-item">
                                    Sections
                        </li>
            </ul>
</nav>


<!-- STOP eZPlatformUIBundle:components:breadcrumbs.html.twig -->


            <h1 class="ez-page-header-name" data-icon="&#xe61a;">Sec
    </header>
        <section class="ez-serverside-content">
        <div class="ez-table-data is-flexible">
            <div class="ez-table-data-container">
                <table class="pure-table pure-table-striped ez-selec
                    <thead>
                        <tr>
                            <th>Section name</th>
```

UI Components

Navigation hub

The Navigation Hub is a side view displaying the top menu.



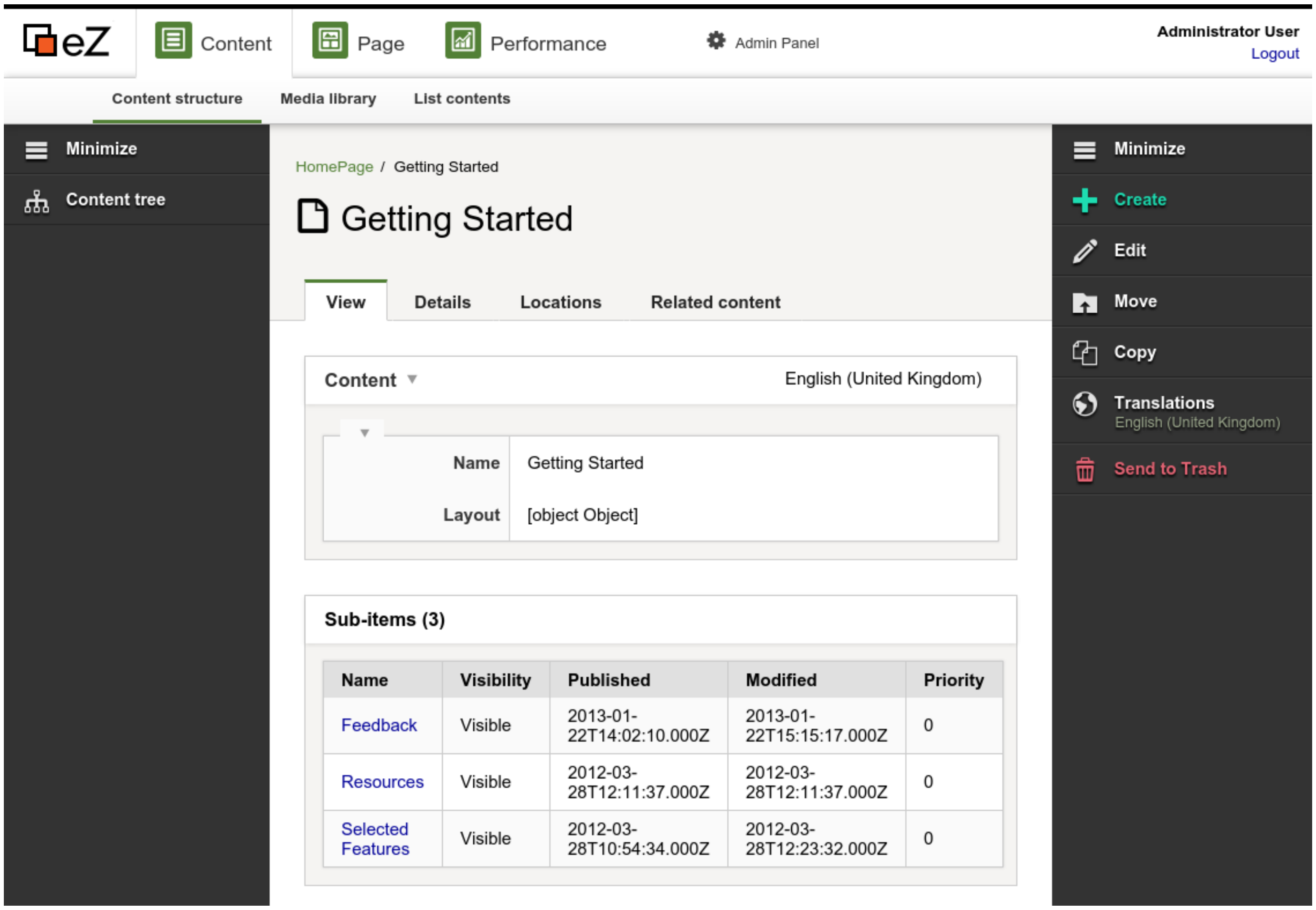
It displays 4 **Navigation zones**:

- *Content*
- *Page*
- *Performance*
- *Admin Panel*

A zone can contain an arbitrary number of **Navigation zone items**. By default, the *Content* zone has 2 navigation items: *Content structure* and *Media library*.

Bar views: Discovery Bar View, Action Bar View, Edit Action Bar View

Bar views provide a set of potential actions for the user.



Name	Getting Started
Layout	[object Object]

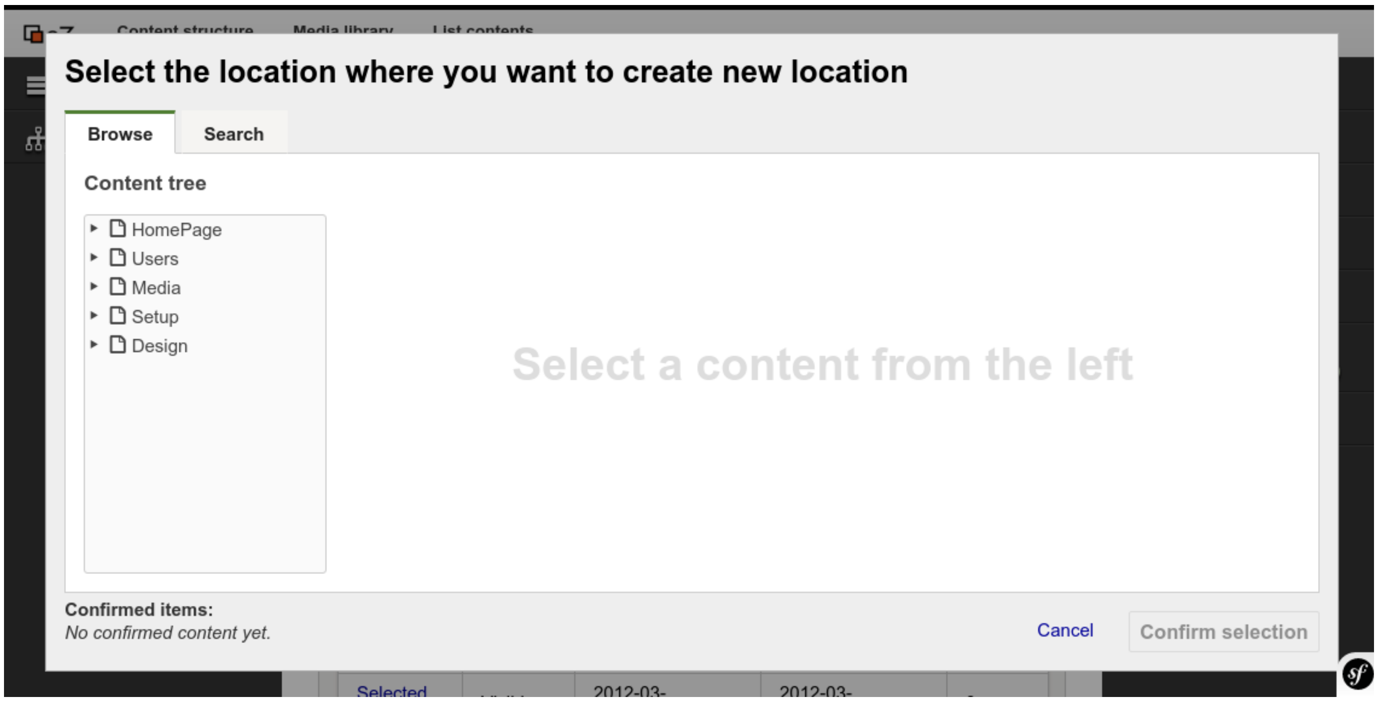
Sub-items (3)

Name	Visibility	Published	Modified	Priority
Feedback	Visible	2013-01-22T14:02:10.000Z	2013-01-22T15:15:17.000Z	0
Resources	Visible	2012-03-28T12:11:37.000Z	2012-03-28T12:11:37.000Z	0
Selected Features	Visible	2012-03-28T10:54:34.000Z	2012-03-28T12:23:32.000Z	0

When navigating in the Content zone, the **Discovery Bar View** allows you to discover content while the **Action Bar View** on the right allows you to act on the Content item being viewed (edit, move, copy, etc.).

When editing a Content item, the **Edit Action Bar View** on the right allows you to act on the Content item being edited.

Universal Discovery Widget



The Universal Discovery Widget is a side view triggered when the user needs to pick a Content item (or a Location). It can provide several **Discovery Methods**. By default, *Browse* and *Search* are available.