(O) JS Backend en vivo

Cobertura

Objetivos

- Para conocer los fundamentos de las capacidades de Backend Live
- Comprenda cómo mejorar AdminCentral para los distintos usuarios
- Comprenda los conceptos de cómo definir su propia API para crear nuevas capacidades de Backend Live.

Abstracto

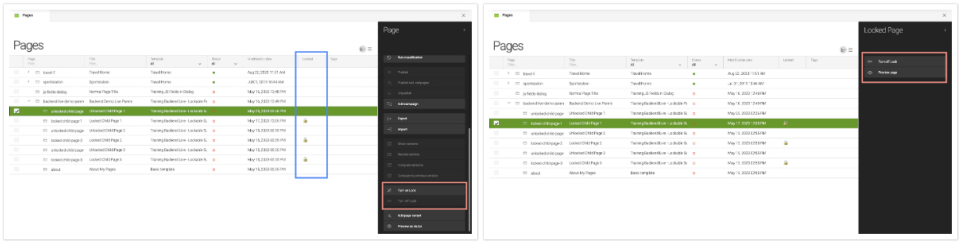
- Proporciona una capacidad de módulo ligero para personalizar una amplia variedad, en realidad abierta, de funcionalidades de backend que anteriormente requerían código Java y reimplementaciones continuas.
- Acciones personalizadas de la subaplicación del navegador
- Reglas de disponibilidad personalizadas
- Definiciones de columnas personalizadas
- Comandos personalizados
- Capacidad para crear sus propias extensiones para reemplazar el código Java backend
- Consulte Backend Live: Extensiones para modelos JavaScript para obtener información más detallada sobre el uso y la configuración.

Conceptos de backend en vivo

- Los módulos Backend Live son una extensión de Javascript Models 2, lo que brinda aún más posibilidades para evitar la necesidad de depender de módulos Magnolia personalizados.
- El proceso de desarrollo se acelera debido al hecho de que está trabajando dentro de módulos de luz, lo que no requiere reiniciar las instancias de Magnolia.
- La implementación se simplifica enormemente ya que solo es necesario actualizar los módulos de iluminación en sus diversos entornos de implementación.
- Las configuraciones de YAML brindan la capacidad de exponer cualquier elemento inyectable en sus scripts de Backend Live

Ejemplos de usos de Backend Live

A continuación se muestra un ejemplo que abarca acciones personalizadas, reglas de disponibilidad personalizadas y una columna personalizada.



El cuadro rojo en la primera imagen representa:

- Dos acciones personalizadas: "Activar bloqueo" y "Desactivar bloqueo"
- Reglas de disponibilidad que permiten a un usuario hacer clic en el elemento solo si una página está bloqueada o desbloqueada

El cuadro rojo en la segunda imagen tiene la misma funcionalidad, pero también muestra otra regla de disponibilidad que no mostrará ninguna otra parte de la barra de acción si la página seleccionada está bloqueada.

El cuadro azul en la primera imagen muestra una definición de columna personalizada que, una vez más, utiliza la propiedad "bloqueada" de una página para mostrar contenido de columna especial, en este caso el

🔒

emoji.

Configuración general de un elemento activo de backend

Este es un ejemplo de una configuración de acción YAML personalizada:

```
toJcrBrowserAppJs: &toJcrBrowserAppActionConfiguration
  label: View in JCR-Browser App  (JS)
  icon: icon-view
  $type: jsAction
  modelPath: /training-backend-js-magic/js-backend-resources/backend-live/actions/OpenSubAppAction.js
  parameters:
    appName: jcr-browser-app
    subAppId: browser
  exposedComponents:
    locationController:
      componentClass: info.magnolia.ui.api.location.LocationController
      name: locationController
```

Una vez que defina **$type: jsAction** , ahora podrá usar la definición de Javascript, lo que le permitirá definir:

- **modelPath** : la ruta del archivo de recursos a su lógica de Javascript
- **parámetros** : las propiedades dinámicas a las que se puede acceder en el archivo Javascript a través del atributo de contexto **de definición**
- **exponenComponents** : un mapa de elementos inyectables que se pueden usar dentro de sus objetos Javascript

Con estos elementos configurados, ahora puede desarrollar su lógica de backend de Javascript. A continuación se muestra un ejemplo de la configuración anterior:

```
var OpenSubAppAction = function () {
    this.execute = function () {
        let pathToOpen = "/";
        const NodeUtil = Java.type("info.magnolia.jcr.util.NodeUtil");
        const NodeTypes = Java.type("info.magnolia.jcr.util.NodeTypes");

        const DefaultLocation = Java.type("info.magnolia.ui.api.location.DefaultLocation");
        const Location = Java.type("info.magnolia.ui.api.location.Location");

        this.logSomething();
        this.logSomethingElse();

        let jcrItem = this.content;
        if (!jcrItem.isNode()) {
            jcrItem = jcrItem.getParent();
        }
        const isContent = NodeUtil.isNodeType(jcrItem, NodeTypes.Content.NAME);
        if (isContent || (this.parameters.containsKey("useContentSubNodes") && this.parameters.get("useContentSubNodes") === true))  {
            pathToOpen = jcrItem.getPath();
        } else {
            pathToOpen = cmsfn.parent(jcrItem, NodeTypes.Content.NAME).getPath();
            this.log.info("Current node is either not content or useContentSubNodes was set to false: ID: {} pathToOpen {}", this.content.getItemId().toString(),
        }

        const addToLocationParameter = this.parameters.get("addToLocationParameter");
```

```
        if (addToLocationParameter != null){
            pathToOpen += this.parameters.get("addToLocationParameter");
        }

        const location = new DefaultLocation(Location.LOCATION_TYPE_APP, this.parameters.get("appName"), this.parameters.get("subAppId"), pathToOpen);

        this.log.info("Moving to {} in the {} app.", pathToOpen, this.parameters.get("appName"));
        locationController.goTo(location);
    }

    this.logSomething = function () {
        this.log.error("I'm something");
    }

    this.logSomethingElse = function () {
        this.log.warn("I'm warning something else");
        this.log.info("One last note")
    }
}

new OpenSubAppAction();
```

Al igual que con el modelo de renderizado de Javascript, debes declarar tu objeto y crear una instancia del mismo:

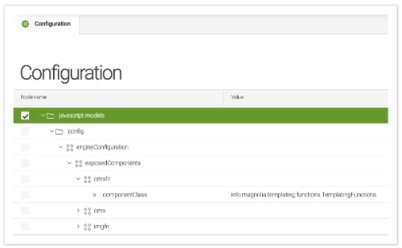
```
// Declare your Javascript object
var OpenSubAppAction = function () {
}
// Instantiate the object
new OpenSubAppAction()
```

> Necesitará usar **var**y no **const**o **let**para que esto funcione correctamente.

Para la configuración de parámetros, se puede acceder a ellos mediante el **this**operador, como cosas como `parameters`, `content`y `log`son objetos de contexto proporcionados al objeto. Los parámetros son un mapa, por lo que debes usar la función Map.get para recuperarlos, es decir`this.parameters.get("appName")`.

Cualquiera `exposedComponents`son atributos de contexto que están expuestos a nivel global y, por lo tanto, no necesitan el operador this. Ahora se puede acceder a cualquier método público expuesto por estos componentes según el nombre proporcionado en la configuración.

También puede exponer atributos de contexto global para todas las implementaciones de Javascript en la aplicación Configuración, utilizando la misma sintaxis que en el archivo de configuración YAML anterior en`/modules/javascript-models/config/engineConfiguration/exposedComponents`



Para obtener acceso a las clases de Java, puede utilizar el método GraalVM para importar una clase mediante la ruta de clase completa. Esto no es lo mismo que incluir un componente expuesto, ya que eso le dará acceso al recurso en memoria de ese elemento inyectable, mientras que esto simplemente le dará acceso a la clase. Puede utilizar cualquier método público de esa clase y también crear instancias de nuevos objetos de esa clase utilizando esta técnica. Aquí hay un ejemplo:

```
        const NodeUtil = Java.type("info.magnolia.jcr.util.NodeUtil");
        const NodeTypes = Java.type("info.magnolia.jcr.util.NodeTypes");

        const DefaultLocation = Java.type("info.magnolia.ui.api.location.DefaultLocation");
        const Location = Java.type("info.magnolia.ui.api.location.Location");

        // Static access to public methods
        const isContent = NodeUtil.isNodeType(jcrItem, NodeTypes.Content.NAME);
        // Using the constructor of DefaultLocation to create a new object
        const location = new DefaultLocation(Location.LOCATION_TYPE_APP, this.parameters.get("appName"), this.parameters.get("subAppId"), pathToOpen);
```