

FirstSpirit Guide:

Developing Modules in the Cloud

Crownpeak Technology GmbH - Version Dec-2022

This document guides you through the module development process within the FirstSpirit Cloud. You will find the general process explained briefly. Links will lead you to the detailed documentation to address specific steps of the journey.

[1. Your FirstSpirit Cloud Environment](#)

[2. Extending FirstSpirit](#)

[2.1 Extending the FirstSpirit ContentCreator](#)

[2.2 Extending the FirstSpirit SiteArchitect](#)

[3. Setting up Your Work Environment](#)

[3.1 Set up access to Git Repositories](#)

[3.2 Setting Artifactory Credentials to Access Module Dependencies](#)

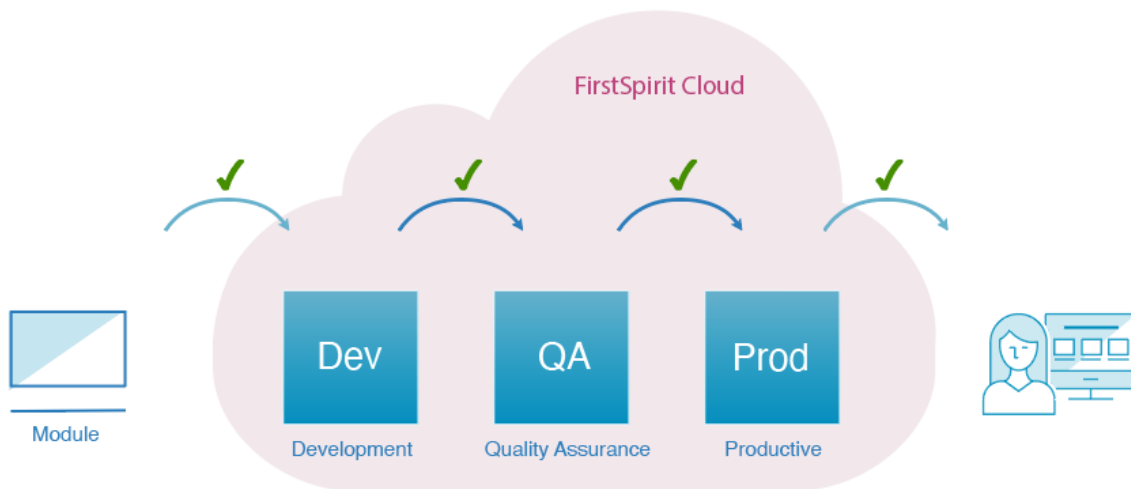
[4. Developing Modules](#)

[5. Distributing Modules](#)

Target audience	(New) module developers within the FirstSpirit cloud setting
Preconditions	DTA (Developer Training Advanced) Training & Coaching Center

1. Your FirstSpirit Cloud Environment

Your FirstSpirit Cloud instance includes three environments: Dev, QA and Prod by default:



Development (Dev)

This environment is where one or more developers can work without posing any risk to each other or the production code.

Quality Assurance (QA)

This environment is the staging or testing area where you can test developed segments to ensure quality before they're transferred to the Production Environment. Therefore this is a nearly exact replica of the production environment.

Production (Prod)

This environment contains all deployed/published projects that may already hold content and are available for your customers.

- All three stages contain nearly the same projects.
- Template and setting changes are regularly moved to the next stage to avoid divergent development states.
- While Dev and QA typically contain test content, Prod contains all of the live project content

Supporting Tools for Module Development

Development Tool	Used in Crownpeak Cloud Development
Programming Language	Java 11
Version Control	Git
Branching Model	Git-Flow
Source Code Repository	Bitbucket
CI/CD	Bamboo
Build Automation Tool	Gradle
Artifact Repository	Artifactory

2. Extending FirstSpirit

FirstSpirit provides multiple extensibility opportunities that allow developers to implement custom server-side as well as client-side functionality. The various extension points of the [FirstSpirit API](#) can be used by various so-called component types.

Components are packaged in and distributed as FirstSpirit modules which are basically ZIP files with the extension .fsm containing all resources and dependencies as well as a module descriptor in XML format.

[Module Definition](#).

Note: All cloud servers use the isolated mode. Some central aspects are described here:

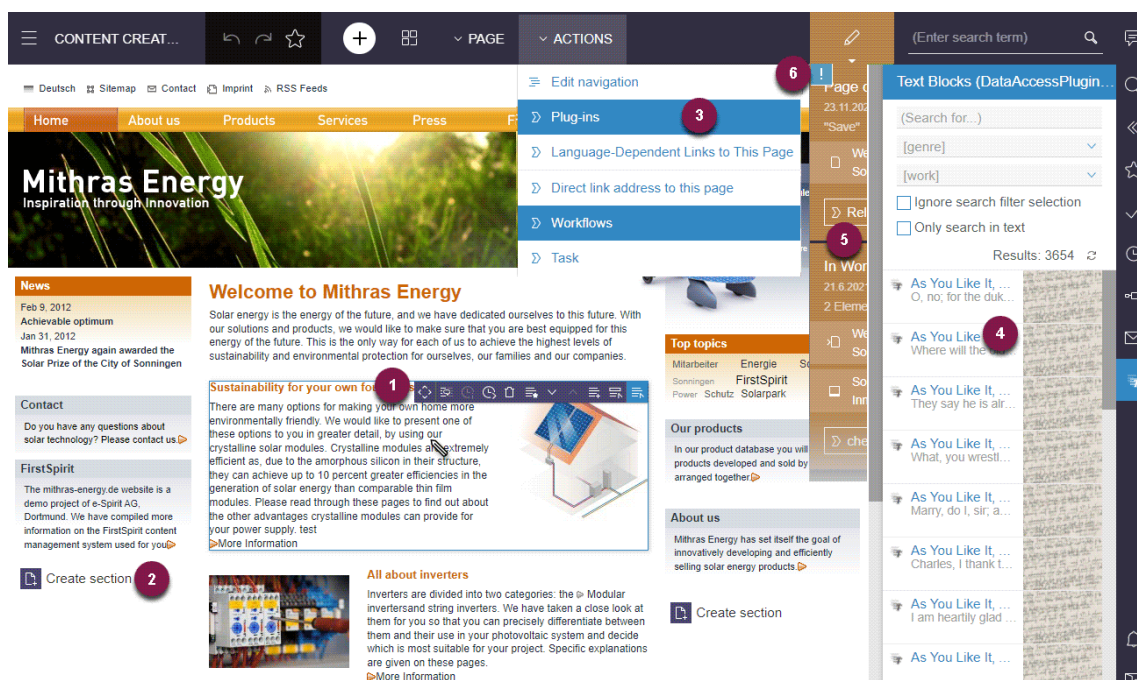
[Module development "Isolated"](#)

2.1 Extending the FirstSpirit ContentCreator

As part of the ContentCreator's user-centered design, Crownpeak provides project developers with means to design custom functionality that presents additional interaction features that will help editors perform specific tasks.

[ContentCreator Extensions](#)

Each of these plug-in types ties into a different area in the ContentCreator user interface and is able to handle FirstSpirit data as well as poll input from users:




1. InlineEdit Items

Insert one or more actions into the button overlay that appears as the mouse pointer hovers over a section, page contents or datasets rendered in the preview pane.

 [InlineEdit Buttons](#)

2. Template Buttons (FS_BUTTON)

Provide access to scripts and Java classes that implement extra functionality. These buttons are placed directly in page, section or dataset templates, may handle click and drag-and-drop interactions, and can be rendered in forms, preview/generated representations of elements or both.

 [Template Buttons](#)

3. Toolbar Menu Items

Specify one or more menu items which will be shown in the Actions menu of the ContentCreator toolbar.

 [Toolbar Menu Items](#)


4. Reports (Data Access)

Integrate external data sources (e.g. web services) with FirstSpirit so that data from these sources (e.g. images, Youtube videos,...) may be referenced within FirstSpirit content.

 [Data Access](#)

5. Element Status and Workflow Grouping

Specify a project-wide algorithm to determine the page status (whether it is considered released, modified or currently in a workflow) that is displayed in the ContentCreator toolbar.

 [Element Status and Workflow Displays](#)

6. Page-Based Notifications

Implement notifications that occur in the ContentCreator preview. They will be shown within the page status flyout of the ContentCreator toolbar as well as - optionally - prominently underneath the page status display.

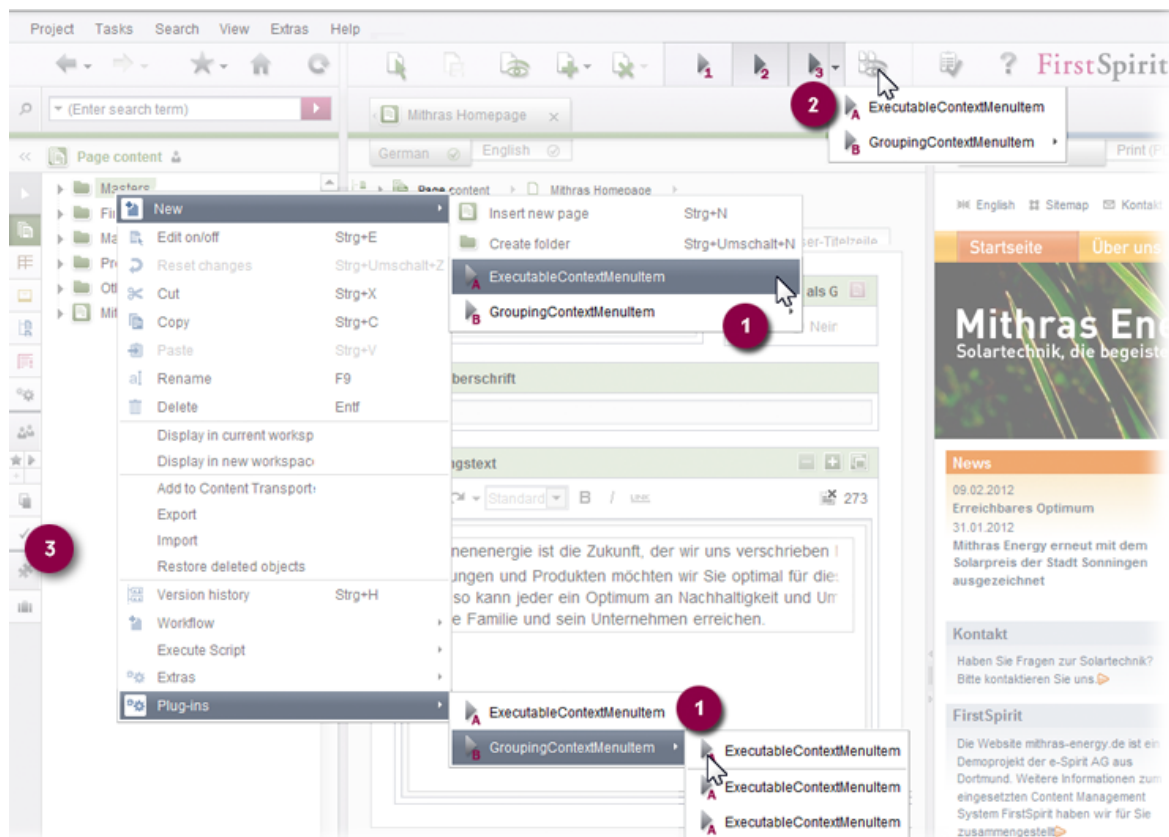
 [Page-Based Notifications](#)

2.2 Extending the FirstSpirit SiteArchitect

The FirstSpirit SiteArchitect provides multiple means to integrate plug-in components. These may implement automation features, additional input components as well as connectors to web services which serve as data providers or further content editing tools.

[SiteArchitect Extensions](#)

Each of these plug-in types ties into a different area in the SiteArchitect user interface and is able to use FirstSpirit API functionality to execute programmatic actions:




1. Context Menu Items

Insert one or more menu items into either the *New* or *Plug-ins* submenus of store element context menus.

 [Context Menu Items](#)

2. Toolbar Items

Insert buttons into the toolbar displayed above the editorial (middle) workspace of the SiteArchitect and work within the context of the FirstSpirit element currently shown in that workspace (e.g. a section whose form is currently displayed).

 [Toolbar Items](#)

3. Reports (Data Access)

Integrate external data sources (e.g. web services) with FirstSpirit so that data from these sources (e.g. images, Youtube videos,...) may be referenced within FirstSpirit content.

 [Data Access](#)

3. Setting up Your Work Environment

As a developer you need to set up your environment. This has only to be done once since the configuration is shared for all repositories. Therefore you will have to

- Set up access to Git repositories
- Set artifactory credentials

3.1 Set up access to Git Repositories

To get access to the Git repository you have to specify your SSH key in your

» [Bitbucket profile](#).

Use an existing key or create an appropriate key first. Add the public key to your profile.

📖 [Official Bitbucket help](#).

3.2 Setting Artifactory Credentials to Access Module Dependencies

Dependencies specified in your module will be downloaded from our Artifactory which acts as a Maven repository. For this to work you need to specify the credentials in your personal, not module-local gradle.properties. 📖 [Gradle Properties](#)

The file is located in `$HOME/.gradle/gradle.properties` and should contain at least these lines:

```
artifactory_hosting_username=CLOUD_USERNAME  
artifactory_hosting_password=CLOUD_ENCRYPTED_PASSWORD
```

`CLOUD_USERNAME` is usually your complete e-mail address. The encrypted password can be retrieved through a simple [Artifactory REST call](#). (If prompted for a username and password, please use your cloud credentials.)

4. Developing Modules

The first step for developing your module is to clone the existing repository from your FirstSpirit cloud git. If the repository does not exist yet, please send a request to the FirstSpirit TechnicalSupport team. After that, please find the newly created repository in [Bitbucket](#).

The following steps may help you developing your own FirstSpirit modules:

Developing modules	Support
<p>1. Send a request to the TechnicalSupport team if the repository does not exist yet. You should use a separate repository for each module.</p>	<p>» Create ticket at TechnicalSupport Please provide the desired (as meaningful as possible) name(s) of the repositories within the ticket.</p>
<p>2. Clone the existing repository in Bitbucket The initial setup already contains a preconfigured Gradle environment. This build automation tool allows for compiling and packaging your module while taking care of things like external dependencies.</p>	<p>» Bitbucket</p> <p>» Gradle Build Tool</p>
<p>3. Packaging a module into an FSM file is performed by the FirstSpirit Module Gradle Plug-in. It creates a module descriptor and adds all required dependencies to the archive.</p>	<p>» FirstSpirit Module Gradle Plug-in</p>
<p>Use a local test server to speed up local development. The FirstSpirit Gradle Plug-in can start such a server for you and install the module automatically. This feature is also useful for automatic integration tests.</p>	
<p>Usage of Web application components Components of web applications can either be used globally or project locally. It's needed to create global components because of resource and performance reasons. Send a request to the Technical Support team to add a global web-app component.</p>	<p>» FirstSpirit web applications</p> <p>» Create ticket at TechnicalSupport Please provide the following information within the ticket:</p> <ul style="list-style-type: none">• name of the instance• name of the component to be added• name of the Web app (e. g. "ContentCreator")• optional: configuration of the component

Checklist for module development:

Compatibility

- Java 11 is used as target level
- The [FSM Dependency Detector](#) does not show any usage of non-API classes

Dependencies

- Checking for newer version (e.g. JUnit, GSON,..)
- Checking for vulnerabilities (e.g. on <https://cve.mitre.org>)
- Large libraries that are used only once and can easily be replaced

Documentation

- README documentation has been added
- A reasonable amount of JavaDoc is present for most classes
- Source code is well commented (?)

Code Style

- Static code analysis detects duplicate code, unused imports, slow String concatenation, inconsistent resource bundles, and other issues
- Logging is performed with FirstSpirit-Logger
- Unnecessary debug logging
- URL concatenation without proper encoding
- Jackson classes should use final fields and appropriate annotations
- Hardcoded URLs
- Logged credentials
- Exception handling with large try blocks and generic "something went wrong"-messages
- Module structure: Swing config classes included in web-app
- Integration tests should be used to test expected API responses and deserialization
- REST calls that create new Apache HTTP clients that are never closed
- Invalid encoding in language resource bundles

Default Module / Feature Requirements

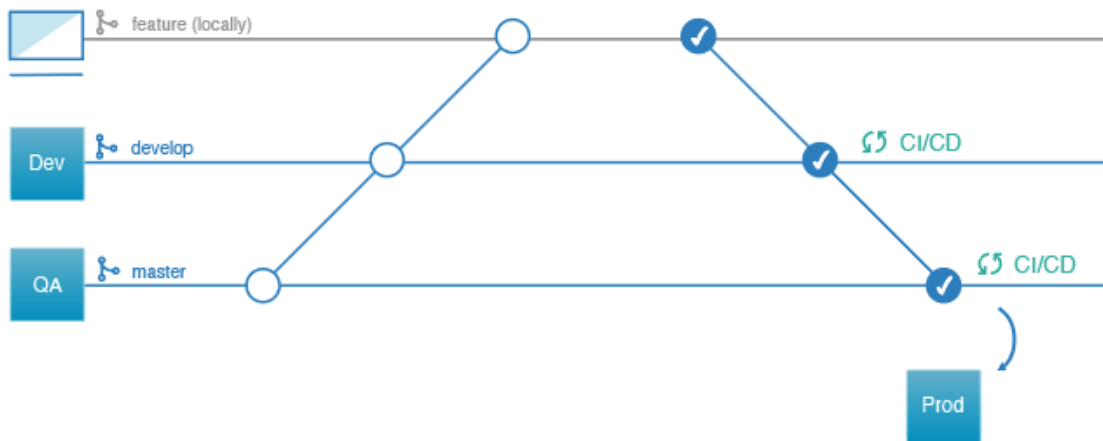
- Module doesn't track its usage
- Module/Component name follows [naming convention](#)
- Web resources are using unique instead of common names (e.g. web/index.js) to prevent them from being overwritten by other components

Usage Test

- Test plan exists
- Project app starts correctly (if the project app does contain invalid configuration, the ContentCreator does not load correctly and only displays a white page instead.)

5. Distributing Modules

Depending on the Git branch you are using, a pushed commit triggers an automatic build process which installs the module in one of your cloud instances. The naming convention is the same as in [Git Flow](#), a Git workflow which we recommend to use.



The following steps may help you developing your own FirstSpirit modules:

Distributing modules

1. Develop features locally

New features should always be developed and tested locally on a feature branch. Pushed commits on these branches will be compiled and tested by our CI pipeline, but your cloud servers will be left unchanged.

Please note that the CI pipeline will fail unless there is at least one successful unit test.

2. Push finished features to the development branch (Dev)

Pushed commits on the branch `develop` will be installed on the dev cloud instance assuming that compilation and test execution did not produce any errors.

» [CI / CD Pipeline](#)

If your module contains WebApp components and your component is deployed the first time you'll have to contact Technical Support to let them add the component to the respective global web application (ContentCreator / Preview).

» [Create ticket at TechnicalSupport](#)

Please provide the following information within the ticket:

- name of the instance
- name of the component to be added
- name of the Web app
(e. g. "ContentCreator")
- optional: configuration of the component

Attention: Updating a module may cause redeployments, e.g. for the ContentCreator. This may interrupt existing user sessions on the development instance.

3. Merge changes to the master branch (QA)

The next step for releasing a new version of your module is the installation on your QA instance. This is performed automatically after merging the changes into the **master** branch.

Note on versioning: On the master branch, the Release plan ensures an automatic increment of the version number of the module (patch number). In order to increase the minor or major version, update the snapshot version number manually before merging the changes to master. We recommend the usage of [Semantic Versioning](#).

Attention: Updating a module may cause redeployments, e.g. for the ContentCreator. This may interrupt existing user sessions on the QA instance.

4. Install your module on the Production instance (Prod)

Installing a new version of your module on the production instance is only possible during a patch-day. Please contact the FirstSpirit TechnicalSupport team for this procedure.

» [Create a ticket at the FirstSpirit TechnicalSupport](#)

Please provide the name of the module and the desired version number within the ticket.