FirstSpirit Guide:
Developing Templates in the Cloud

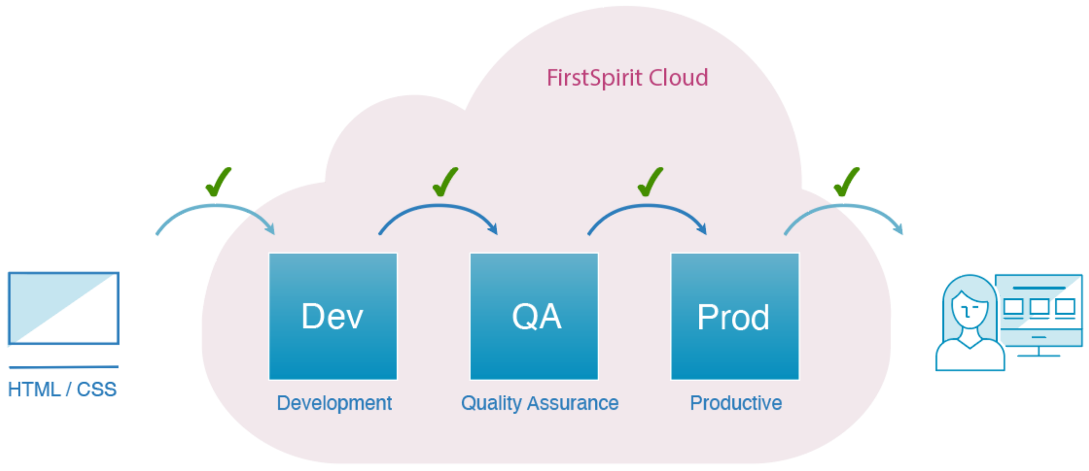
Crownpeak Technology GmbH - Version Dec-2022

This document guides you through the template development process in 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.

Target audience	(New) template developers within the FirstSpirit cloud setting
Preconditions	DTB (Developer Training Basic) Training & Coaching Center

1. Your FirstSpirit Cloud Environment

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



Development (Dev)
This stage is where one or more developers can work without posing any risk to each other or the production code.

Quality Assurance (QA)
This stage is the staging or testing area where you can test developed segments to ensure quality before they're transferred to the Production stage.

Production (Prod)
This stage contains all deployed/published projects that may already hold content and are available for your customers.

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

Preparations - Setting up the environment

≫ Creating a new Project
≫ Adding a new user
≫ Signing in as developer

2. Developing templates

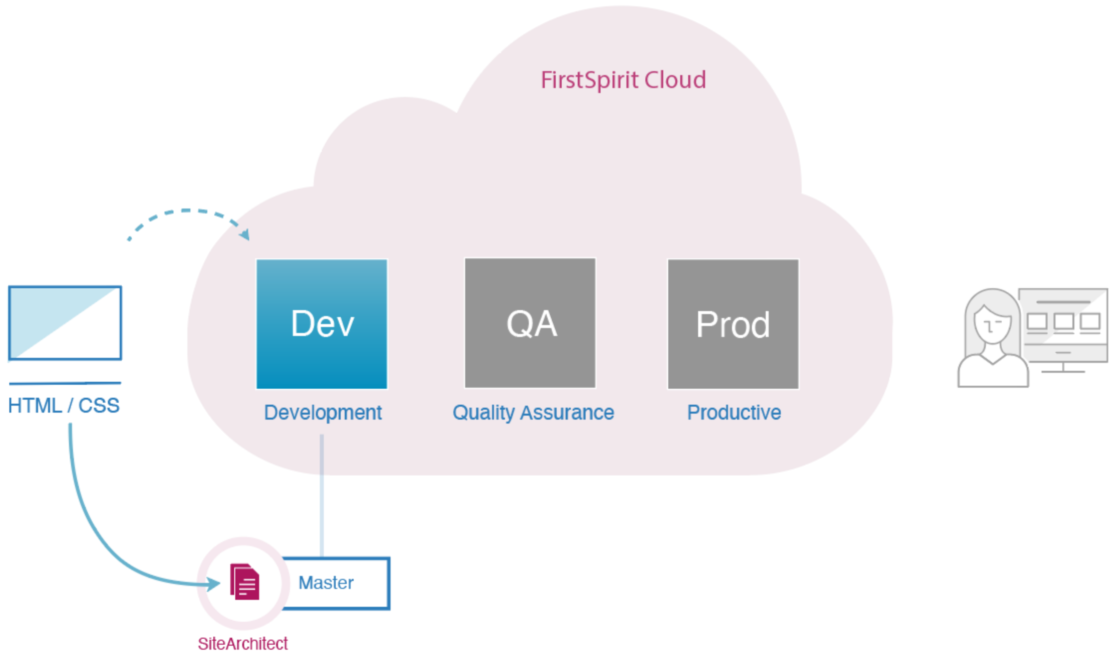
There are two ways to develop your project:

Developing a classic HTML project

In a classic HTML project the HTML, CSS and JavaScript are developed in advance and transferred to your project in the FirstSpirit SiteArchitect. You might be using this project type when your editors are going to work directly within the preview of the concrete target website.

Developing a headless project

In a headless project editors are only working with content which is distributed into FirstSpirit CaaS (Content as a Service) - a data pool of which the content can be accessed on different channels (PWA, touchpoint, …). You might be using this project type if you are developing a modern application with separation of front-end and content or diverse applications which will be using the same content in different target applications.



a) Developing a classic HTML project

Content model and front-end (HTML, CSS, JS) are fully created and maintained in FirstSpirit SiteAchitect.

Developing a classic HTML project	Support
1. Starting with an empty project	The basics of template development Fundamentals about FirstSpirit template development
A. Manually creating templates according to the HTML code structure (HTML may be provided by an agency)	Composition of templates
B. Importing templates Use the HTML importer tool	The Template Wizard
2. Creating **forms** for data entry by editors	Creating forms / The "form" tab
How should data be **output** and your website look like?	FirstSpirit template syntax
3. Uploading **images and files** (incl. CSS and JS)	The FirstSpirit Media Store
4. Checking and **debugging** templates	Debugging template errors
Further capabilities of FirstSpirit templating and syntax reference (Scripting, API, guiding the editor etc.)	FirstSpirit template development
A. Continuously developing in an **external** tool (Recommended for major design updates.)	The FirstSpirit Media Store Upload updated CSS and JS files to theMedia Store Composition of templates Transfer modifications in HTML to the respective template in SiteArchitect
B. Continuously developing in the **SiteArchitect** (Recommended for smaller modifications.)	Composition of templates Accomplish smaller changes in HTML directly in SiteArchitect

b) Developing a headless project

The front-end is usually hosted externally (e.g. PWA in Git), and the content model is maintained in FirstSpirit SiteAchitect.

Developing a headless project:	Support
1. Developing the **front-end** You can use any **PWA** as front-end, no FirstSpirit output channel needed. Examples: Custom apps (e.g. built with Angular, React, Vue.js), front-ends as a Service (e.g. Vue Storefront, Frontastic), Crownpeak blueprint custom app (FirstSpirit Experience Accelerator)	The FirstSpirit FSXA FirstSpirit Experience Accelerator as an exemplary project
2. Developing the content model The content is filled in FirstSpirit by means of **forms.**	Creating forms / The "form" tab
Starting an **e-commerce** project In e-commerce projects, content and structure are defined by the FirstSpirit project, the front-end is developed separately. The frontend often strongly depends on the used e-com system.	FirstSpirit Connect for Commerce - Reference Project on Git Our reference project allows an easy start for template development in e-commerce environments.
Retrieving data in the front-end Data is provided by FirstSpirit in a standardized, not adaptable JSON format. It is automatically delivered to FirstSpirit CaaS and can be retrieved by the front-end.	Caas product documentation Fundamentals about FirstSpirit CaaS CaaS Connect for Administrators How to configure CaaS Connect FirstSpirit's standardized JSON output How does the JSON output look like? Navigation Service How to build dynamic navigations for headless scenarios

3. Distributing templates

Why to use template distribution

- **Support multi-stage development process** (develop, test, release).
  Sharing work results across different stages (Dev, QA, Prod).
- **Synchronize multiple sub-projects** (master → sub-project)
  Sharing work results across different projects.
  Sub-projects can be projects for different countries of your web site (for example EN, DE, FR…). Use your "master" project for template development and distribute them to sub-projects for ensuring consistent design and structures across all projects, if needed.

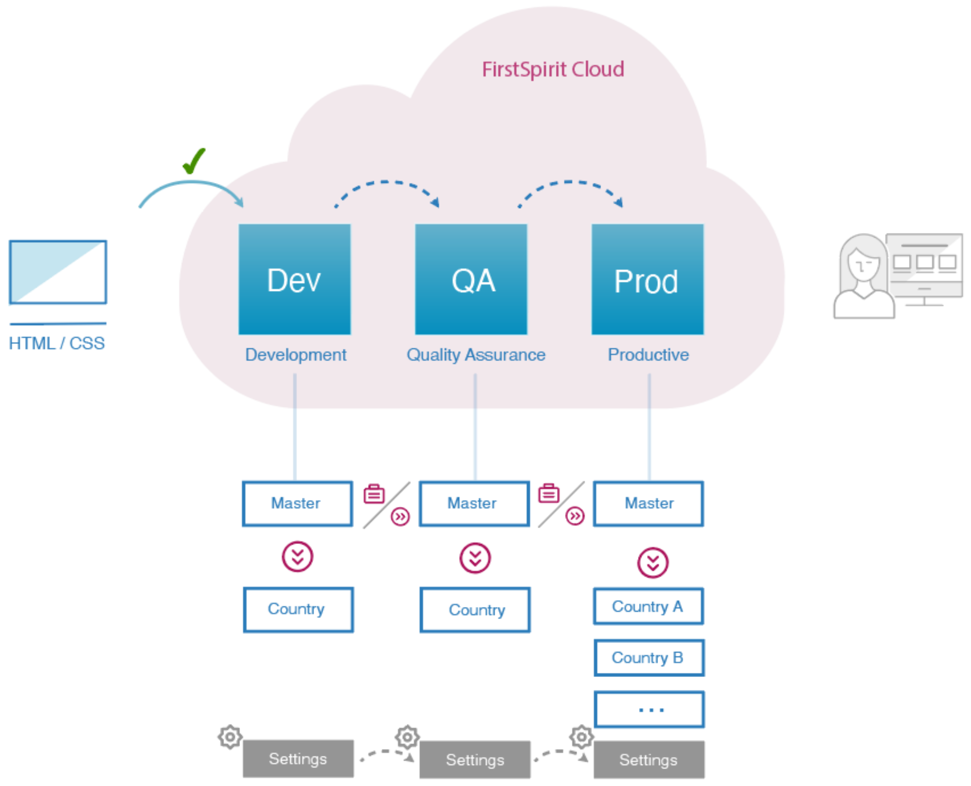
FirstSpirit offers different tools for distributing templates.

Template distribution

Module for simply distributing all templates, mainly between sub-projects. In the default setting, only templates and technical media are distributed with a few clicks, across the 3 stages (Dev, QA, Prod). The capabilities can be enhanced by defining a custom configuration.

Content Transport

Capability for distributing selected templates, mainly between development stages. It allows a more custom configuration, for example: only modified templates. Moreover, it visualizes dependencies to referenced elements to make it easier for you to determine and transport complete structures.



a) Distributing templates via stages: Dev → QA → Prod

Sharing templates with different stages:	Support
1. Transporting templates **Dev → QA** Transport the templates of your master project on Dev to your master on QA	
a) using "Template distribution" if you would like to easily transport **all templates** at once	[What is Template distribution and how to use it]
b) using 'Content Transport' if you would like to transport only a **subset of templates**	[Content Transport]
● Transfer templates via **FS-CLI** using the **feature download** and **feature install** commands	[FS-CLI Releases] See command reference bundled with FS-CLI for more info.
● Use the **Filesystem feature storage** and transport templates via the "/opt/firstspirit5/s3-transfer" directory. (AWS S3 Bucket present in the Dev, QA, and Prod environments)	[Content Transport] See **Section 4.9** (Configuring the storage locations)
2. **QA:** check the result	
3. Transporting templates **QA → Prod** Transport the templates of your master project on QA to your master-project on Prod	
a) using "Template distribution" if you would like to easily transport **all templates** at once	[What is Template distribution and how to use it]
b) using 'Content Transport' if you would like to transport only a **subset of templates**	[Content Transport]
● Transfer templates via **FS-CLI** using the **feature download** and **feature install** commands	[FS-CLI Releases] See command reference bundled with FS-CLI for more info.

● Use the **Filesystem feature storage** and transport templates via the “/opt/firstspirit5/s3-transfer” directory. (AWS S3 Bucket present in the Dev, QA, and Prod environments)	Content Transport See **Section 4.9** (Configuring the storage locations)
4. **Prod:** check the result	
Transporting templates to **sub-projects** If using sub-projects, include them into your transport strategy (see a).	
Continuous development takes place in the master project ≫ transport changes regularly to the next stage ≫ check result on the next stage ≫ include potential sub-projects to your transport strategy (see a). ≫ only transport templates of your **master** to the next stage	

b) Master → Sub-project

Transporting templates (master → sub-project)	Support
1. Transporting your **templates** Transport your templates from the master to the sub-project	
a) using "Template distribution" if you would like to easily transport **all templates** at once	What is Template distribution and how to use it
b) using 'Content Transport' if you would like to transport only a **subset of templates**	Content Transport
● Transfer templates via **FS-CLI** using the **feature download** and **feature install** commands	FS-CLI Releases See command reference bundled with FS-CLI for more info.
● Use the **Filesystem feature storage** and transport templates via the "/opt/firstspirit5/s3-transfer" directory. (AWS S3 Bucket present in the Dev, QA, and Prod environments)	Content Transport See **Section 4.9** (Configuring the storage locations)
2. Adjusting **settings** in the sub-project (languages, resolutions, ContentCreator settings...)	Set project properties
3. **Checking** the result Check the results in the sub-project (layout, function,...)	Using version comparison
Continuous development takes place in master project - transport changes regularly to the sub-project using "Template distribution" and check the result in the sub-project (layout, function...)	

4. Tutorial For Distributed Development of FirstSpirit Project **[BETA]**

This tutorial presents a concrete example of developing a demo FirstSpirit project in a distributed fashion using git and FS-CLI.

4.1 Concept

The **Git-based development process** associates a FirstSpirit project with a Git repository, that contains all the design elements like templates and technical media (javascript libraries, css files, etc.).

The Git repository helps to track the developed features, which can make it easier to identify and resolve issues that may arise later on.

The process also supports **feature-branches** in Git that allows multiple team members to work on different requirements or bug fixes simultaneously and in isolation from the main project or other developments. For each Git branch a **separate FirstSpirit branch-project** exists that helps to easily develop and test changes, merge them when they are ready, without disrupting the main project.

We use ExternalSync to synchronize between the Git repositories and FirstSpirit projects, and ContentTransport to enhance new branch-projects with example content for a better development experience as well as to transport the content between D/Q/P stages.

Overall this process improves the **efficiency** and **collaboration** in the team and improves the quality of the codebase in an effective way.

4.2 Setup of Git and FS-CLI

Initial setup	Support
1. **Install FS-CLI**	📖 The "FSDevTools" command line tool
1.1 **Define FS CLI variables** 1.2 **Add the executable directory** of the FS-CLI (fs-cli/bin) to your system's PATH variable so that you can run the FS-CLI from the project directory.	*Find example values below.*
2. **Install git**	📖 Using version control systems (VCS)
2.1. **Configure git**	📖 Configuring external components
3. **Test** FS-CLI can connect to the FS server	`>fs-cli.cmd -e test`

FS-CLI Variable	Example value	Description
`fshost`	ecs-dev-caas-eu-central-1-1.e-spirit.hosting	FS server host
`fsport`	443	FS server port
`fsmode`	HTTPS	connection mode
`fsuser`	username@my-company.com	username
`fspwd`	Passw0rd!	password

4.3 Set up a project (initial transfer to an empty git)

Setting up a project (empty git)	CLI Command / Support

1. **Create a git repository** for the project	`>mkdir first-spirit-tutorial`
	`>cd first-spirit-tutorial`
	`>git init .`

2. **Import** a FirstSpirit **demo project**	
2.1 **Download the archive** with a FirstSpirit demo project	📖 GitHub (ExternalSync) 📖 GitHub (Project Export)
2.2 **Import the project** into your FS server	`>fs-cli.cmd -e project import --importProjectName "Demo SmartLiving Project (master)" --importProjectDescription "Demo SmartLiving Project (master)" --projectFile "D:\path\to\smartliving-updated.tar.gz" --databaseLayerMapping "*:FirstSpiritDBA"`

3. **Create .gitignore** and **add** the following **content** to it	`>.FirstSpirit`
	`>/lastCommandResult.json`
	`>/*.log`
	`>/*.log.gz`

4. **Commit the .gitignore** to git	`>git add .gitignore`
	`>git commit -m "JIRAPROJECT-12345 add .gitignore"`

5. **Export the project** via external sync into the file system	`>fs-cli.cmd -e --project "Demo SmartLiving Project (master)" --syncDir "./fs-projects/demo-smartliving-project" export --permissionMode ALL -- projectproperty:ALL contentstore globalstore pagestore sitestore templatestore mediastore`

4. **Add** the exported files to git, **commit and push** them
The master git branch then contains the "Demo SmartLiving Project (master)" FS project.
	`>git add fs-projects`
	`>git commit -m "JIRAPROJECT-1234 synced FS project and added it to git"`
	`>git remote add origin [your git repository url]`
	`>git push -u origin master`

4.4 Use-Case: Implementing a requirement (#42)

Implementing a requirement	Support
1. **Create** a new git **feature branch** for later use This step represents the work of another developer working on the same project and implementing other features. This Branch will be used in [4.5 Use-Case: Implementing a requirement with conflicts (#23)](#)	`>git checkout -b JIRAPROJECT-42_my_new_feature`
2. **Import FS project** from the file system into a new FS project on the FS server using external sync → The local git branch JIRAPROJECT-42_my_new_feature now has the FS project "Demo SmartLiving Project (JIRAPROJECT-42_my_new_feature)" as a companion FS project.	`>fs-cli.cmd -e --project "Demo SmartLiving Project (JIRAPROJECT-42_my_new_feature)" --syncDir "./fs-projects/demo-smartliving-project" import --permissionMode ALL --layerMapping "*:FirstSpiritDBA"`
3. **Export** the newly created companion project back into the file system using external sync → This should change some of the project's files.	`>fs-cli.cmd -e --project "Demo SmartLiving Project (JIRAPROJECT-42_my_new_feature)" --syncDir "./fs-projects/demo-smartliving-project" export --permissionMode ALL --projectproperty:ALL contentstore globalstore pagestore sitestore templatestore mediastore`

4. **Commit and push** these changes	`>git add fs-projects`
	`>git commit -m "JIRAPROJECT-42 changes due to re-sync"`
	`>git push -u origin JIRAPROJECT-42_my_new_feature`

5. **Implement** the new **feature** using SiteArchitect	📖 [Developing templates](#)
6. **Sync the project** into the file system	`>fs-cli.cmd -e --project "Demo SmartLiving Project (JIRAPROJECT-42_my_new_feature)" --syncDir "./fs-projects/demo-smartliving-projec`

```
t" export --permissionMode ALL --
projectproperty:ALL contentstore
globalstore pagestore sitestore
templatestore mediastore
```

7. **Add** the implementation of the new **feature to git**
Among the changes you'll notice e.g. that the "fs-projects/demo-smartliving-project/TemplateStore/PageTemplates/standard/FS_References.txt" file has been changed.

```
>git add fs-projects
>git commit -m "JIRAPROJECT-42
implemented my new feature"
>git push
```

8. **Check** the **demo project**
Open "Demo SmartLiving Project (JIRAPROJECT-42_my_new_feature)" FS project and check that everything is fine

9. **Create a merge request**
After the merge request review it can be merged by fast forwarding because the master branch has been merged into the topic branch.

10. **Resync** the **FirstSpirit master project**
This brings the new feature in the master branch and in the master FS Project.

```
>fs-cli.cmd -e --project "Demo
SmartLiving Project (master)"
--syncDir
"./fs-projects/demo-smartliving-projec
t" import --permissionMode ALL
--layerMapping "*:FirstSpiritDBA"
--import-comment "upd after merge"
```

11. Delete branches

11.1 **Change** back **to master branch**

```
>git checkout master
```

11.2 **Delete** the **FS project branch**
The feature branch FS project may now be deleted (e.g. via ServerManager).

```
>fs-cli project delete
--deleteProjectName "Demo SmartLiving
Project
(JIRAPROJECT-42_my_new_feature)"
```

11.3 **Delete** the **git feature branch**
The feature git branch may be deleted as well

```
>git branch -D
JIRAPROJECT-42_my_new_feature
>git push --delete origin
JIRAPROJECT-42_my_new_feature
```

4.5 Use-Case: Implementing a requirement with conflicts (#23)

Implementing a requirement with conflicts	CLI Command to perform
1. **Checkout feature branch** for merge conflict simulation	`>git checkout -b JIRAPROJECT-23_my_new_feature`
2. **Import a FirstSpirit project** from the file system into a new project on the FirstSpirit server using external sync This creates a FirstSpirit project "Demo SmartLiving Project (JIRAPROJECT-32_my_new_feature)" as a companion FS project in the local git branch JIRAPROJECT-23_my_new_feature	`>fs-cli.cmd -e --project "Demo SmartLiving Project (JIRAPROJECT-23_my_new_feature)" --syncDir "./fs-projects/demo-smartliving-project" import --permissionMode ALL --layerMapping "*:FirstSpiritDBA" --import-comment "initialize project for git feature branch"`
3. **PERFORM STEPS 3-7** in Chapter 4.4 (Export to the file system, commit & push changes, implement the new feature, sync the project, add the project to git)	Chapter 4.4
4. **Fetch latest git changes**	`>git fetch --all --prune --tags`
5. **Prepare feature branch** for a merge request Since we have already merged a new feature into master in the step 4.4, we expect a merge conflict when merging with master.	`>git merge origin/master`
6. **Checking parallel work** Suppose the colleague developer has also added a new format template with the same name and UID, then check status.	`>git status`
7. **Resolve merge conflict** Resolve Git merge conflicts manually in a diff tool and then commit the corrected versions to Git.	

There are some cases where we need to resolve the merge conflict manually. Examples are UID clashes, where we would rollback the merge and rename the conflicting template.

8. **Resync** the project back into the file system

```
>fs-cli.cmd -e --project "Demo
SmartLiving Project
(JIRAPROJECT-23_my_new_feature)"
--syncDir
"./fs-projects/demo-smartliving-projec
t" export --permissionMode ALL --
projectproperty:ALL contentstore
globalstore pagestore sitestore
templatestore mediastore
```

9. **Commit** the changes to git

```
>git rm -r
fs-projects\demo-smartliving-project\T
emplateStore\FormatTemplates\content_f
ormat_templates\format_template_for_my
_new_feature\
>git add fs-projects
>git commit -m "JIRAPROJECT-23 rename
conflicting format template‟
>git push
```

10. **Try merge** master branch again
This time less conflicts should show up.

```
>git merge origin/master
```

11. **Sync project** from file system into companion FS project after merge conflict resolution

```
>fs-cli.cmd -e --project "Demo
SmartLiving Project
(JIRAPROJECT-23_my_new_feature)"
--syncDir
"./fs-projects/demo-smartliving-projec
t" import --permissionMode ALL
--layerMapping "*:FirstSpiritDBA"
--import-comment "upd after merge"
```

12. **PERFORM STEPS 8-11** in Chapter 4.4 — 📖 Chapter 4.4