Modularity

modularity, frontend, backend, architecture, installation

TODO: Introduction

Backend

After discussing the topic, Spring Plugin was chosen as a framework for ensuring backend modularity. Compared to other considered frameworks such as OSGi or Microservices, it is the simplest option based on interfaces, their implementations and registration of these implementations for their use in the application. The accepted disadvantage is application server restart after installing a module (module = **.jar package**).

Installation/ module update

Installation/ module update is done by copying the module into the application libraries **<idm.war>/WEB-INF/lib**. Modules can also be included directly in the project dependencies (pom.xml modul app).

Module descriptor - ModuleDescriptor

Each module installed in the application must have its descriptor - **ModuleDescriptor**. The descriptor contains the metadata about the module:

- unique identifier of the module (the module is referred to by the identifier everywhere).
- module version
- whether the module can be removed / disabled - some modules are essential for the application (core, the ecosystem itself ...)
- descriptive metadata: name, description, provider
- permissions added within the module
- ...
- in the future, the information about dependent modules in specific versions will be added in the modules, e.g. module acc version 1.3.8 will be dependent on the core version 1.0.0 to 1.5.0, and the like

All modules - their descriptors - are registered at the start of the application and can be managed through service **ModuleService** providing mainly:

* the list of installed modules * the list of permitted modules * enabling/ disabling of the module - linked to configuration service **IdmConfigurationService**. By convention, item **idm.pub.<identifikátor_modulu>.enabled** is used for determining of an enabled / disabled module. This item gains boolean value.

Database scripts

Each module can contain its own database scripts.

Example module

The example module was created for module demonstration and, at the same time, as a template (skeleton) for creating a module with pre-filled configurations. It contains:

* **pom.xml** - dependency settings * **module descriptor** - descriptor * **flyway configuration** - for db initialization

If the example is used for creating a new module, the key word **example** has to be replaced with another unique module identifier and then various descriptions (module name, description, etc.) can be added. For future uses, it is planned to create a maven archetype or other module skeleton generator.

Enabling / disabling of module

An agenda which shows the list of installed modules and enables enabling / disabling of modules which support this function (see descriptor) was created on the frontend.

After disabling of a module, all the services (e.g. rest) must become unavailable. This is done by annotation IfEnabled, which, after adding it over a service or an individual method (definition over a service has a higher priority), ensures a control over the enabled module or configuration item before calling a service / method itself. Calling a service of a disabled module ends in exception **ModuleDisabledException** or rather **ConfigurationDisabledException**.

If a frontend module is linked to / dependent on a backend module, it is also disabled.

TODO:

* checking of module id uniqueness at the start of application * maven archetype for generating the module skeleton

Frontend

The goal of frontend modularity was to fulfill the following scenario:

1. A standard version of CzechIdM (e.g. 7.0) is installed on the frontend server.
2. The administrator downloads a czechidm (npm) module, which he loads in folder czechidm-modules (in module czechidm-app).
3. The administrator runs a compilation (command **gulp build see more in the installation manual**)
4. By doing that, a new module was registered in the application and its content is displayed to the user in the application (including integration of individual menus, user desktops, etc.)

Application building

Building of a frontend application including its modules is done by the **Gulp** building tool. The definition of the building process is save in file **gulpfile.babel.js**. This file contains a series of tasks which are run in a fixed order during building:

- **clean** - Deletes previous build (folder /dist)
- **makeModules** - Crates symlinks to node_modules of the application for all modules found in "czechidm-modules" folder, (as shown in Project Structure chapter).
- **loadModules** - Finds all "module-descriptor.js" in node_modules of the application (thanks to the symlinks finds modules in czechidm-modules as well). For each module found, it generates a line for adding a link to the module descriptor, where the module ID is used as the key (will be used in moduleAssembler).
- **createModuleAssembler** - Creates a resulting moduleAssembler.js which will contain links to all the module descriptors of the installed modules. The default **moduleAssembler.js** is in the czechidm-app/src module. It is possible to add a descriptor into the default moduleAssembler for modules different from the ones created at building. Adding the loaded modules is done by replacing the compile mark (<compile mark>) in the default assembler with lines generated in the previous task. The resulting moduleAssembler.js is created in folder "/dist".
- **loadModuleStyles** - Loading of styles from individual modules. The llocation of the main less file is defined in the module descriptor of the correspondent module in the item mainStyleFile' (e.g. 'src/css/main.less'). The location of all the main styles is saved in 'paths.srcIncludedLess' for further processing
- **loadModuleRoutes** - Finds the definition file route for the given module. Route is a definition of the url and the content which should be used for it. The path to the main file defining module routes can be found in the module descriptor in item "mainRouteFile". It will generate a line for adding a link to the definition for each route definition found, where the module ID is used as the key.
- **createRouteAssembler** - Creates a resulting routeAssembler.js which will contain links to all the installed modules' routes. The default routeAssembler.js is located in the czechidm-app/src module. It is possible to add a route into the default moduleAssembler for modules different from the ones created at building. Adding the loaded modules is done by replacing the compile mark (<compile mark>) in the default assembler with lines generated in the previous task. The resulting moduleAssembler.js is created in folder "/dist".
- **loadModuleComponents** - Finds the definition file of the components for the given module. A component is some content from the module which can be used in other modules thanks to a definition in the component descriptor. The path to the main file defining module components can be found in the module descriptor in item "mainComponentDescriptorFile". It will generate a line for adding a link to the definition for each component definition found, where the module ID is used as the key.
- **createComponentAssembler** - Creates a resulting routeAssembler.js which will contain links to all the installed modules' components. The default componentAssembler.js is located in the czechidm-app/src module. It is possible to add a route into the default componentAssembler for modules different from the ones created at building. Adding the loaded modules is done by replacing the compile mark (<compile mark>) in the default assembler with lines generated in the previous task. The resulting componentAssembler.js is created in folder "/dist".
- **themes** This task is responsible for the correct selection of the application theme. The main theme (the path to it) is defined in the configuration under "theme", for example "czechidm-core/themes/default", which means that, in this case, this theme is located in the czechidm-core

module in the themes/default folder. All image (in '/images/') will be copied into the "/dist" folder. All the less styles (in '/css/') will be added into the variable 'paths.srcIncludedLess' (it will be processed in the next tasks).
- **runTest** - Runs all the mocha tests in all the installed modules (finds all the tests in the folder /test recursively).
- **config** - Generates the resulting application configuration. The selection of configuration is done on the basis of input parameters (stage, profile) at compilation execution. The resulting configuration is saved in the "/dist" folder.
- **urlConfig** - Generates a configuration file for an external change in selected application parameters. It takes over 'serverUrl' from the main configuration. The resulting file is saved in the "/dist" folder.
- **styles** - It run a compilation of less styles. The main less style is saved in 'czechidm-app/src/css/main.less'. Imports of other less styles acquired in previous tasks are included in this file (styles of individual modules, main theme styles). The resulting css is saved in the "/dist" folder.
- **loadModuleLocales** Copies all the locales located on the path which is defined in the module descriptor in the 'mainLocalePath' item to the "/dist" folder.
- **browserify** - Generates a resulting javascript 'app.js' containing all the javascripts from the used modules and saves it in the "/dist" folder.

Project structure

Below is the structure of the project and its modules. The folder containing all npm modules "node_modules" (in the main module czechidm-app) is designed as a **symlink**. Its actual location is one level higher which was necessary to prevent an error with more copies of ReactJS (see https://facebook.github.io/react/warnings/refs-must-have-owner.html).

This error occurs in a situation when one installation of the React npm module is in 'czechidm-app/node_modules' and the other in its submodule 'czechidm-app/node_modules/czechidm-core/node_modules'. In this case the error described above occurs.

A solution is to remove the duplicate installation from React from the submodule. If this submodule isn't really located under the file czechidm-app but for example next to it (in reality, it is in this way in the repository), then czechidm-core will find no installation of React. The reason is that **NPM** looks for modules in node_modules and if it doesn't find them, it moves on to a higher level recursively. Because of that, the main node_modules folder is (**and must be**) located above all czechidm modules (app, core, acc, etc.)

```
/project
└──node_modules
|    |
|    └──~czechidm-core (symlink to czechidm-modules/czechidm-core)
|    |
|    └──~czechidm-example (symlink to czechidm-modules/czechidm-example)
|
└──czechidm-app
|    |
|    └──~node_modules (symlink to parent node_modules)
|    |
```

```
|     └──dist (final compiled source for web server)
|     |
|     └──czechidm-modules
|         └──czechidm-core
|         |     |
|         |     └──node_modules
|         |     |
|         |     └──src
|         |     |     |
|         |     |     └──css
|         |     |     |  main.less
|         |     |     |
|         |     |     └──locales
|         |     |        cs.json
|         |     |
|         |     └──themes
|         |     |  module_descriptor.js
|         |     |  routes.js
|         |     |  component_descriptor.js
|         |     |  package.json
|         |
|         └──czechidm-example
|         |     |
|         |     └──node_modules
|...            ...
```

Project structure (developer mode)

The structure is almost the same as in the previous "production" example. The difference is in the location of submodules, which aren't under the 'czechidm-app' folder but next to it (real location in the repository). Symlinks to these submodules are placed in 'czechidm-app/czechidm-modules/' for that reason.

```
/project
└──node_modules
|     |
|     └──~czechidm-core (symlink to czechidm-modules/czechidm-core)
|     |
|     └──~czechidm-example (symlink to czechidm-modules/czechidm-example)
|
└──czechidm-app
|     |
|     └──~node_modules (symlink to parent node_modules)
|     |
|     └──dist (final compiled source for web server)
|     |
|     └──czechidm-modules
|         |
|         └──~czechidm-core (symlink to folder with czechidm-core module)
|         |
```

```
|        └──~czechidm-example (symlink to folder with czechidm-example
module)
|
└──czechidm-core
|    |
|    └──node_modules
|    |
|    └──src
|    |    |
|    |    └──css
|    |    |  main.less
|    |    |
|    |    └──locales
|    |       cs.json
|    |
|    └──themes
|    |  module_descriptor.js
|    |  routes.js
|    |  component_descriptor.js
|    |  package.json
|
└──czechidm-example
|    |
|    └──node_modules
....
```