Dynamic UI Fragment Composition

The microservice architecture primarily addresses the distribution of responsibility within the project. Considering *loosely coupling* and *high cohesion* also the introduction of a central user interface (UI) is not expediently. Implicitly, the diverse small project teams are responsible for their microservices including the user interface.

Next to the challenge of the distribution of responsibility, also the basic principles of *Human-Computer-Interaction* (HCI) have to be considered and implemented. Furthermore, the use case of a coherent organization like Viva con Agua forces to consider a *Corporate Design* (CD) Besides non-functional requirements, the microservices have to follow informal behaviors as described in this concept. Moreover, new social procedures and protocols have to ensure the alignment of the microservice implementations to the sketched concept.

Solution

The Heureka! architecture is implemented as *Rich-Internet-Application (RIA)* to address the requirements of the *Pool*[2]. It will be possible to use the system on the desktop, but also on a mobile device in the browser. Thus, client technologies are limited to HTML, CSS and JavaScript.

The sketched solution bases on two columns:

- Shared CSS
- Widgets

The shared CSS allows to jointly use design elements and layout descriptions. Therefore, it enables a common CD.

Widgets support the reuse of UI elements by various services. So, many services will need to select users using the UI. Users are managed by the microservice *Drops* and thus, also the UI elements handling users should be implemented by *Drops*. Next to maintainance issues, also a CD and the consistence of user experiences will be supported by the concept of widgets. Furthermore, the architecture considering widgets directly support the concepts of *loosely coupling* and *high cohesion* regarding the UI.

General and global elements, like the navigation or central, static content (impress or header) can become implemented as widgets.

Shared CSS

Global CSS is implemented and delivered by the microservice *Dispenser* that mainly focusses non-functional requirements. Thus, the CSS can be integrated by other microservices by just adding the reference to the global CSS file.

The CSS follows the guidelines regarding modular CSS (https://css-tricks.com/css-style-guides/ and http://cssguidelin.es/), works as a pattern library and is implemented in LESS (http://lesscss.org/) to support maintainance and further development.

Widgets

The microservices can use widgets to provide functions including the user interface. Thus, other microservices are able to integrate these functions with only a minimal effort. Widgets can be integrated just by using a URI specific to the widget. The concept follows the idea of *transclusions*. The providing microservices are the source for the URIs and by a `GET` call, the user interface element is delivered. These user interface elements are not just static content, but implement dynamic behavior.

Next to the standard behavior of HTML elements, also JavaScript implementations can be used. The styling can become implemented by using the global CSS as well as styled (scoped) tags specific to the widget.

The delivered widget always implements HTML code extended by CSS and JavaScript. Since widgets will often require some parameterization and may have return values or side effects, take a look at the How-To of implementing widgets.

Furthermore, every widget requires a documentation describing the parameterization, the behavior, possible return values and side effects. Supporting the decentral and loosely coupled management of the project, widgets have to be deployed using semantic versioning (http://semver.org/).