# OASP Angular Architecture

This document defines how the Angular (v2) framework implements the OASP Client Reference Architecture mapping its concepts to those of OASP. Since Angular 2 provides often multiple ways of implementing them, this document also defines the preferred and consistent way for realization of the required components..

We assume Angular CLI and webpack to be used when developing such applications.

## Mapping to OASP Client Architecture

### Angular Architecture Summary

The architecture of Angular is summarized very thoroughly here:

<https://angular.io/docs/ts/latest/guide/architecture.html>

As an orientation the following figure of that page is repeated here:



Figure Angular Architcture (Source: s. link above)

### High Level Archicture

|  |  |
| --- | --- |
| Component | Implementation in Angular |
| Dialog Component | Each dialog is developed as an Angular component already supporting the idea of presentation and core layer by the use of templates and component class (s. below). |
| Dialog Container | The dialog container is provided by the Angular framework itself represented by the root module. It already covers most of the concepts of the reference architecture natively. |
| Application Server Proxy | Each REST service provided by an OASP4J server should be represented by a concrete Typescript interface marked as an injectable Angular service. CobiGen is to be used for generation of these interfaces. The complete application server access component should be bundled in one NgModule providing access to the complete service layer of the application server. |

### Dialog Container

|  |  |
| --- | --- |
| Component | Implementation in Angular |
| Application | The Angular root component represents the application component covering the bootstrapping and the Angular framework. |
| Configuration Management | An Environment Constant class will be provided for each environment. This is part of webpack and allows us to define environment-specific variables, which are bundled during the build for the concrete environment. |
| Service Registry | The Angular Injector natively implements this component. It allows to define services which are wired to the components during initialization. |
| Dialog Management | Again covered by the Angular root module defining all dialog components in the declaration array also allowing nesting these components within each other. |

### Dialog Components

Each dialog component corresponds to one Angular component (s. above). The presentation layer is implemented by the component’s template and the dialog core by the component class.

#### Presentation Layer

|  |  |
| --- | --- |
| Component | Implementation in Angular |
| View | HTML template of the angular component |
| Presentation data | Data nodes of the DOM tree. |
| Presentation state | Attribute nodes of the DOM tree steering the visualization |
| Presentation events | DOM events |
| Data Binding | Native angular support by using:   * Two-way data binding for all user input * Interpolation or property binding for read-only values (e.g. dynamic titles) |
| Action Binding | Native angular support by using event binding. |
| State Binding | Native angular support by using two-way data binding |

#### Dialog Core Layer

|  |  |
| --- | --- |
| Component | Implementation in Angular |
| Data Storage | Own typescript class generated from transfer objects via CobiGen. |
| Action Processing | Dialog core represented by the class of the Angular component. |
| Dialog state | Own typescript class generated from transfer objects via CobiGen. |
| Interaction of dialog components | Angular already natively supports all required interactions as described in more detail here: <https://angular.io/docs/ts/latest/cookbook/component-communication.html> |
| Embedding | Dialog components can be nested by simply specifying their selector in the parent view. Data can be passed and returned by Input and Output parameters. In addition each dialog core can access its children via the ViewChild decorator. |
| Dialog Flow | The angular router implements the flow between dialog components. |

### Other concepts

Angular 2 provides many more concepts, not directly required by our client architecture:

* Directives
* Modules
* …?

TODO: define in more detail how they relate to our architecture and how they should be used.

## Angular Usage Conventions and Best Practices

### Dialog Component

For very simple dialog inlining the template dialog is permitted. For medium to complex dialogs, an explicit template HTML file is required.

### Service Registry

Angular allows an hierarchical dependency injection out of the box even with the possibility to abstract from the concrete implementation being wired:

<https://angular.io/docs/ts/latest/guide/dependency-injection.html>

Note however, that Typescript interfaces cannot be used to define the service abstraction, since they are not present during runtime.

TODO: define concrete usage convention

### Data Binding

TODO: when to use interpolation and when property binding?

### State Binding

TODO

### Action Binding

TODO

## Packaging Conventions

TODO:

* One big project for client vs. one project per dialog component
* How do NgModules relate to dialog components? 1:1 or 1:n relation?
* One NgModule per component on the server side?
* One NgModule per project or multiple?

## Directory Conventions

* We follow the webpack conventions for an angular project.
* Introduce directories for layers or dialogs?