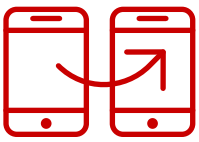
The Consummate Component Checklist for Developers

So you're thinking about creating components? That's great! Follow these practices to ensure great experiences and to foster reusability.

API Design

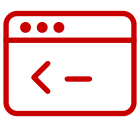
- [ ] **Focus on the main use cases**
  Implement 20% of the features that cover 80% of the use cases.
- [ ] **Limit the number of parameters and offer great default values**
  Ensure that your parameters only cover the most common use cases. Evaluate if your use case requires mandatory parameters. If there are optional parameters, offer the best default values for them.
- [ ] **Use an advanced option to cover other use cases**
  Use JSON as an extensibility mechanism for the extra parameters not covered in the main use cases. List the extra options in the JSON parameter description.

LifeCycle Events (Mobile)



- [ ] **Consider the scenarios where you should define variable default values with the OnInitialize event**
  Prevent poor user experiences when navigating to the previous screen (back navigation).
- [ ] **Prevent memory leaks with the OnDestroy event**
  Ensure that everything that is initialized (for example, variables and event listeners) is destroyed.
- [ ] **Use the OnRender event for components depending on DOM rendering**
  Define a handler for the OnRender event to ensure that you only manipulate DOM elements when they are ready.

CSS and JavaScript



- [ ] **Use style classes and avoid the inline style attribute**
  Take advantage of expressions in style class properties (mobile) or the class attribute (web) instead of using the "style" attribute to improve performance and simplify customization.
- [ ] **Allow customization by avoiding specific CSS styles**
  Avoid the *!important* CSS tag, since using it will override custom CSS. Also, avoid using styles and elements that aren't easily overridden.
- [ ] **Write CSS Selectors in JavaScript related to a runtime ID**
  Implement CSS selectors, in JavaScript, specific to a unique instance by using its runtime ID to enhance reusability instead of using only classes that would affect several elements.
- [ ] **Allow extensibility through JavaScript APIs**
  Provide a set of JavaScript functions so others can extend component functionality. Ensure access to created JavaScript Objects.

Accessible and Easy Code

- [ ] **Add notes to complex code**
  Write a simple and brief description explaining the code that is visible in an action flow. This is key to making it easy to review and improve.

- [ ] **Add comments inside Expression Editors and JavaScript Nodes**
  Use comprehensive comments to describe specific and important elements of the code that might not be easily understood.

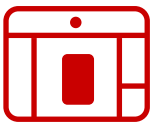
- [ ] **Add labels to all nodes**
  Provide good labels to ensure that everyone can understand the functionality of every node by simply reading them. This will improve the readability of the application flow.

Visual Cues: Names, Descriptions and Icons

- [ ] **Give meaningful and PascalCase names**
  Apply names and descriptions to applications, modules, screens, blocks, variables, actions, events and placeholders. This will help people understand the purpose of your component and how to use it.

- [ ] **Set all icons for readability**
  Apply the same icon, or variations, to applications, modules, blocks and actions. Other developers will see the icons in the development environment and relate them to your component.

Preview: the Developer Experience



- [ ] **Use *If* widget for good previews**
  Set the condition to *False* and add your code to the *False* branch. Create a good preview in the *True* branch of the *If* widget.

- [ ] **Use Service Studio's exclusive CSS tags to improve preview**
  Adapt the preview during development time by using the -servicestudio- tag.

- [ ] **Place sample content inside placeholders**
  If content is mandatory for a component to work as expected, add default content to its placeholders.

Non-Functional Requirements

- [ ] **Test performance to peak conditions**
  Set up a scenario as close as possible to a real one and use the peak number of records or conditions to see how the component performs.

- [ ] **Ensure scalability by supporting multiple blocks**
  Test multiple blocks in the same screen as well as concurrency scenarios. Note: This only applies to blocks.

- [ ] **Implement security mechanisms in your component**
  Consider ciphering local storage data and implement server-side validation.