SDC & Form API, better late than never

Florent Torregrosa

Grimreaper

Drupaler since 2011 (D6), professional since 2013. Tech expert at Smile

Community contributions:

- UI Suite ecosystem since 2021
- Active maintainer of 30+ projects
- Contrib and core patches
- French translation moderator
- Events co-organizer
- Former French Drupal association board member

Current situation

Photo by ELEVATE

Form API , since Fall 2005 (Drupal 4)

Nested arrays, describing a form structure.

Declarative. Easy to type. Easy nesting. Easy alteration.

Return value of:

- `FormInterface::buildForm()`
- `PluginFormInterface::buildConfigurationForm()`
- `WidgetInterface::settingsform()`
- `WidgetInterface::formElement()`
- …

```
return [
  'phone_number' => [
     '#type' => 'tel',
     '#title' => t('Your phone
number'),
  ],
  'actions' => [
     '#type' => 'actions',
     'submit' => [
       '#type' => 'submit',
       '#value' => t('Save'),
       '#button_type' => 'primary',
     ]
  ]
];
```

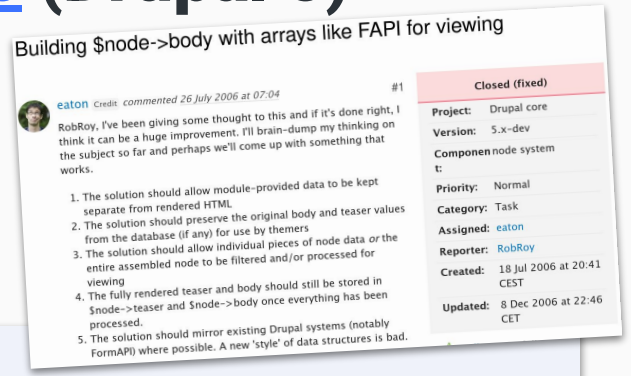
Render API , since Summer 2006 **(Drupal 5)**

Inspired by the Form API.

Return value of:

- `LayoutInterface::build()`
- `FormatterInterface::view()`
- `FormatterInterface::viewElements()`
- `BlockPluginInterface::build()`
- …

You can put (most) render elements in a form.


Building $node->body with arrays like FAPI for viewing

eaton Credit commented 26 July 2006 at 07:04 #1

RobRoy, I've been giving some thought to this and if it's done right, I think it can be a huge improvement. I'll brain-dump my thinking on the subject so far and perhaps we'll come up with something that works.

1. The solution should allow module-provided data to be kept separate from rendered HTML
2. The solution should preserve the original body and teaser values from the database (if any) for use by themers
3. The solution should allow individual pieces of node data *or* the entire assembled node to be filtered and/or processed for viewing
4. The fully rendered teaser and body should still be stored in $node->teaser and $node->body once everything has been processed.
5. The solution should mirror existing Drupal systems (notably FormAPI) where possible. A new 'style' of data structures is bad.

Closed (fixed)

Project:	Drupal core
Version:	5.x-dev
Component:	node system
Priority:	Normal
Category:	Task
Assigned:	eaton
Reporter:	RobRoy
Created:	18 Jul 2006 at 20:41 CEST
Updated:	8 Dec 2006 at 22:46 CET

```
return [
  "#theme" => "item_list",
  "#items" => [
    [ "value" =>
      [
        "#theme" => "image",
        "#uri" => "/path/to/image"
      ]
    ]
  ]
]
```

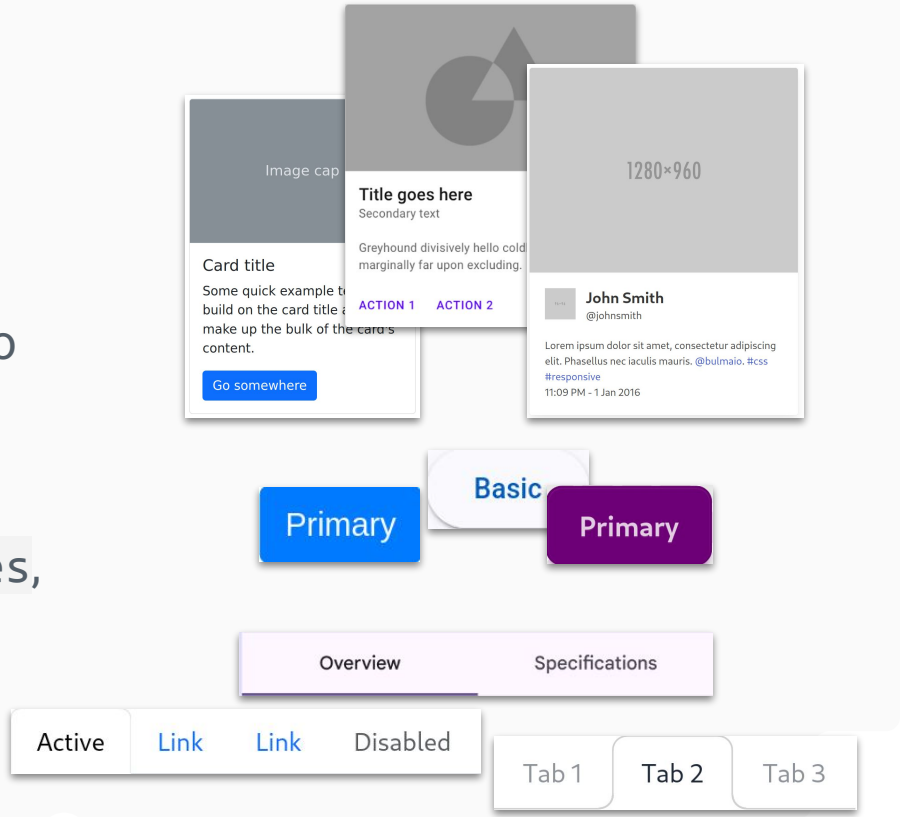
SDC, since Spring 2023 (Drupal 10)

The *front devs friendly* Render API.

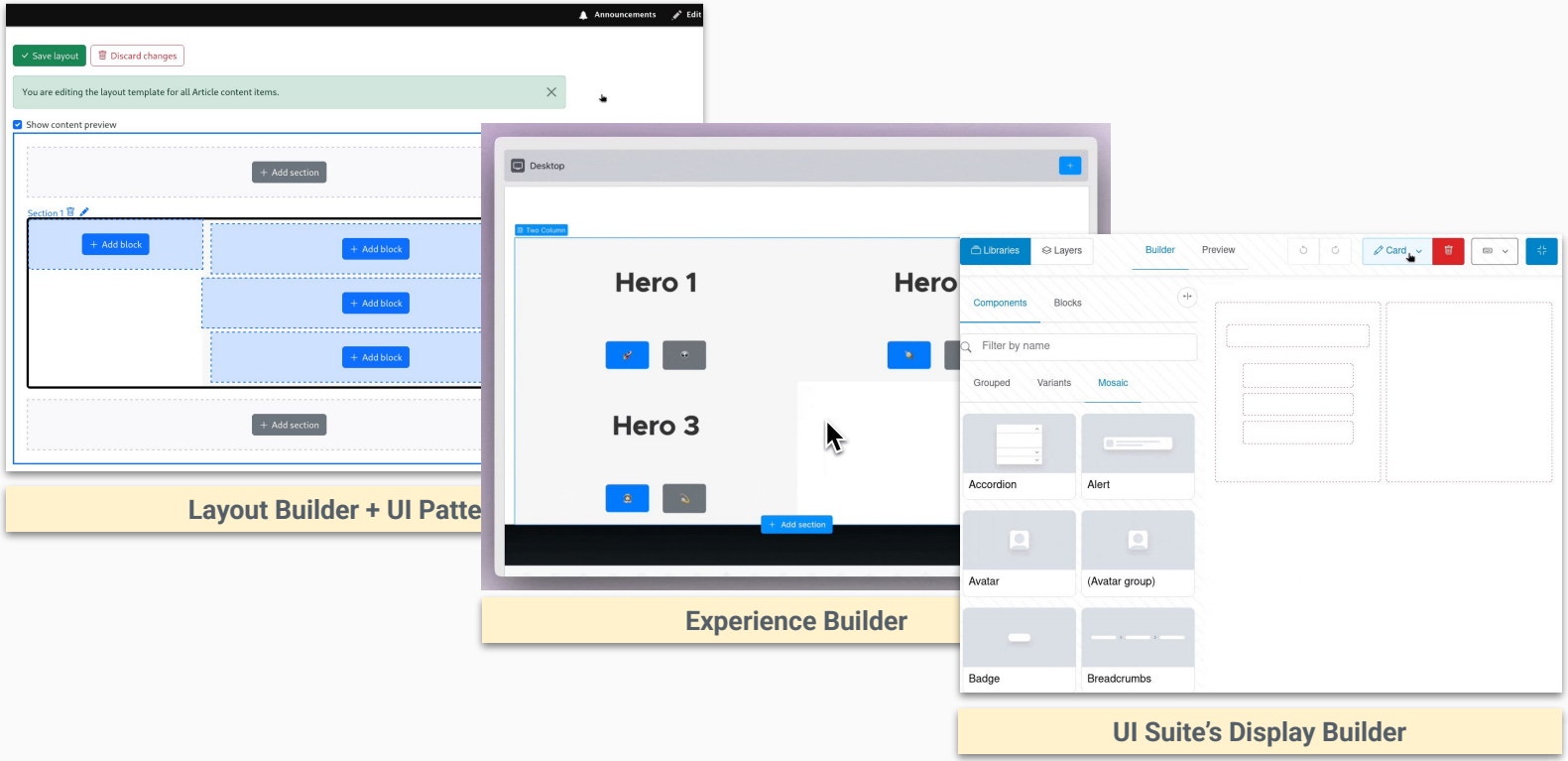
No PHP needed. UI focused & reusable components.

Good stuff inside: Twig templating, plugin discovery, render element, no preprocesses…

SDC can replace most of render elements (`table`, `status_messages`, `pager`…) and hook themes (`breadcrumb`, `progress_bar`, `links`…).



SDC is changing the display building



Problem 1: Using SDC in a Form

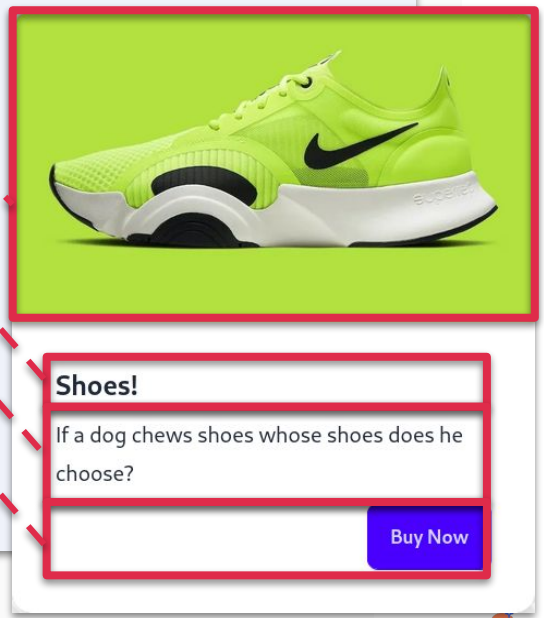
Photo by Tembela Bohle

Components are made of slots and props

Slots: “areas” for free renderables only, like other components.

Props: strictly typed data only, for some UI logic in the template.

```
name: Card
description: "Cards are used to group and..."
group: "Data display"
slots:
  image:
      title: Image
  title:
      title: Title
  text:
      title: Text
  actions:
      title: Actions
props:
  type: object
  properties:
      heading_level: {}
      image_bottom: {}
```



Renderables are made of children & properties

Properties: Prefixed with “#”, data used in the render logic.

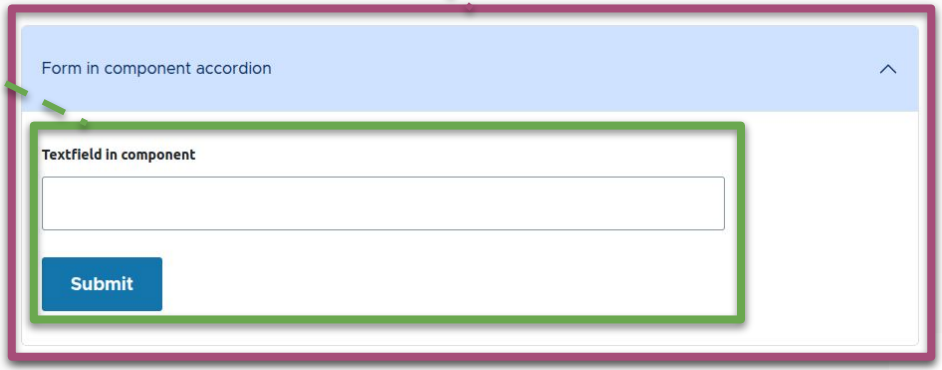
Children: Not prefixed, free renderables.

```
$build['my_details'] = [
  '#type' => 'details',
  '#title' => 'Example',
  '#open' => TRUE,
  'content' => [
    '#markup' => 'Example content',
  ],
];
```

Does it ring a bell?

✅ We can insert a form inside a component slot

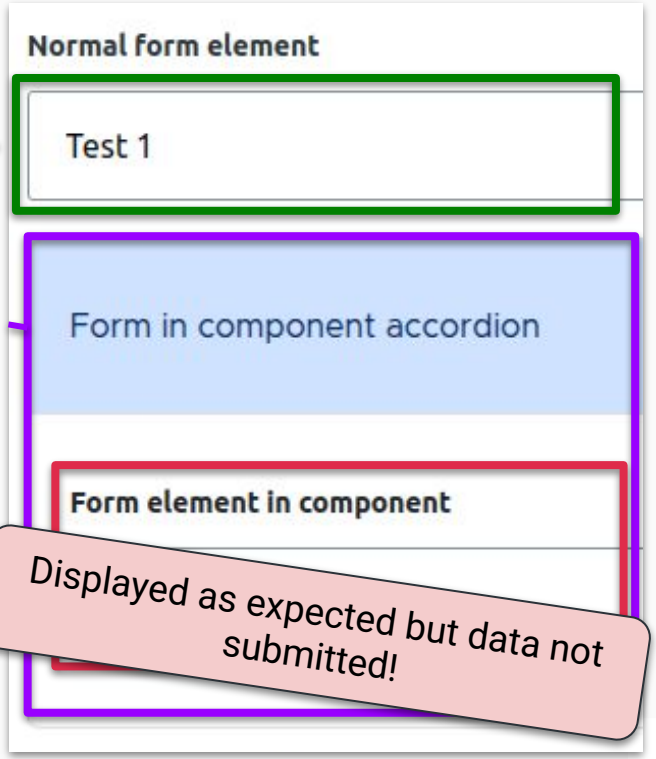
```php
return [
  '#type' => 'component',
  '#component' => 'core_sdc_form:accordion',
  '#slots' => [
    'title' => 'Form in component accordion',
    'content' => $this->formBuilder()->getForm(...),
  ]
];
```



❌ But we can't insert a component in a form

```php
$form['normal'] = [
  '#type' => 'textfield',
  '#title' => 'Normal form element',
];
$form['component'] = [
  '#type' => 'component',
  '#component' => 'core_sdc_form:accordion',
  '#slots' => [
    'title' => 'Form in component accordion',
    'content' => [
      'form_element_in_component' => [
        '#type' => 'textfield',
        '#title' => 'Form element in
component',
      ],
    ],
  ],
];
```

FormBuilder::doBuildForm() expects children, not #slots property.



Step 1: slots as render element children

```
$form['component'] =  [
  '#type' => 'component',
  '#component' => 'foo:accordion',
  '#slots' => [
    'title' => 'Accordion title',
    'content' => [
      'form_element_in_component' => [
        '#type' => 'textfield',
        '#title' => 'Hi there',
      ],
    ],
  ],
];
```

```
$form['component'] =  [
  '#type' => 'component',
  '#component' => 'foo:accordion',
  'title' => 'Accordion title',
  'content' => [
    'form_element_in_component' => [
      '#type' => 'textfield',
      '#title' => 'Hi there',
    ],
  ],
];
```

Step 1: slots as render element children

In ComponentElement::preRenderComponent(), let's put the children (already processed by the form builder) into the slots.

```php
// Handle children as slots.
$children = Element::children($element, TRUE);
foreach ($children as $key) {
  $element['#slots'][$key] = $element[$key];
}
```

✅ Form API is able to go through the render tree.

✅ Component element has its slots.

Step 2: Let's leverage #name property

We still don't get the values when submitting the form.

Mismatch between slots hierarchy and names generated by FormBuilder from #parents

Let's use #name explicitly instead

```
'content' => [
  '#type' => 'component',
  '#component' => 'core_sdc_form:card_body',
  'content' => [
    [
      '#type' => 'textfield',
      '#title' => $this->t( string: 'Textfield in content'),
      '#name' => 'content',
    ],
    [
      '#type' => 'select',
      '#title' => $this->t( string: 'Select in content'),
      '#name' => 'content_select',
      '#options' => [
        'option_1' => $this->t( string: 'Option 1'),
        'option_2' => $this->t( string: 'Option 2'),
      ],
      '#empty_option' => $this->t( string: 'Empty option'),
    ],
  ],
],
```

Step 2: Let's leverage #name property

In FormBuilder::handleInputElement()

```
$input = NestedArray::getValue($form_state->getUserInput(), $element['#parents'], $input_exists);
if (!empty($element['#name'])) {
  $input = NestedArray::getValue($form_state->getUserInput(), [$element['#name']], $input_exists);
}
else {
  $input = NestedArray::getValue($form_state->getUserInput(), $element['#parents'], $input_exists);
}
```

In FormStateValuesTrait::setValueForElement()

```
public function setValueForElement(array $element, $value) {
  if (!empty($element['#name'])) {
    return $this->setValue($element['#name'], $value);
  }

  return $this->setValue($element['#parents'], $value);
```


Problem 2:
We can't define form element with SDC
Photo by ELEVATE

Why is it important?

Form elements are (long) PHP classes instead of Twig + YAML.

It keeps the front dev away and are hard to manage.

Let’s move those to the theming scope.

However, SDC components are stateless.

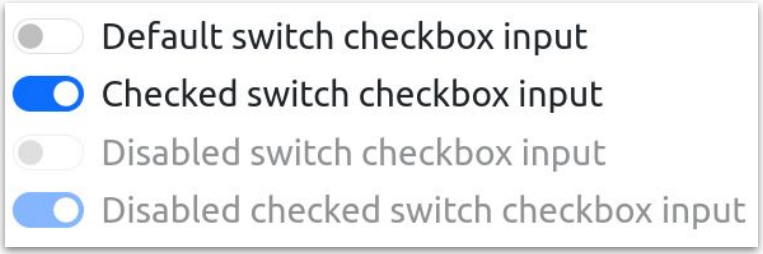
The missing components

In a design system, (around) 20% of components are form elements.

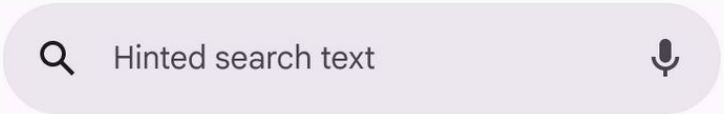
Checkboxes, date pickers, sliders, input groups…

Today, they are skipped by the SDC authors.

Example: Bootstrap Switches



Example: Material Search



Step 1: Change in Render pipeline

A form element is a render element with 2 extra render properties:

- `#input = True`, added by ElementInfoManager if the element implements FormElementInterface
- `#name`, added by the form builder from the parent when missing

So:

- ComponentElement must implements FormElementInterface
- We can leverage the `#name` property expectations already discussed

From an user point of view

A SDC form element is a SDC renderable with #name

Example:

```
[
  "#type" => "component",
  "#component" => "ui_suite_bootstrap:date",
  "#name" => "foo",
  "#default_value" => "2025-04-15",
  "#slots" => [],
  "#props" => [],
]
```

Step 2: Twig template additions

We inject a new variable with data needed for form processing:

- `form_state.name` (string)
- `form_state.value` (mixed)
- `form_state.required` (boolean)

More may be added later.

```
{% set id = id|default('mytextfield-' ~
random()) %}
{% set input_attributes =
create_attribute({
 type: 'text',
 name: form_state.name,
 id: id,
 value: form_state.value,
}) %}
<div{{ attributes.addClass('mb-3') }}>
 <label class="form-label" for="{{ id }}">
   {{ label }}
 </label>
 <input{{ input_attributes }}>
</div>
```

Input or wrapper template?

In general a Drupal form element uses 2 templates:

- The wrapper (`form-element.html.twig` for half of them) with label, errors & description
- The input (`input.html.twig`, `textarea.html.twig`, `select.html.twig`…)

After studying design systems (Bootstrap, Material, DSFR…), we decided to target wrappers.

But still challengeable.

What will be the benefits?

Benefit 1: Form API simplification

37 form elements in Core

4

with only applicative logic (no rendering)

Item, LanguageSelect, Submit & Value

❌ Not suitable for SDC

26

Do we split them?

- UI logic (`#theme`, `#theme_wrapper` & `#pre_render`) to SDC
- App logic (`#process`, `::valueCallback()` & `#element_validate`) kept in PHP classes

7

without applicative logic

Button, Date, Hidden, ImageButton, Search, Tel & VerticalTabs

✅ Convertible to SDC

Benefit 2: Front dev empowerment

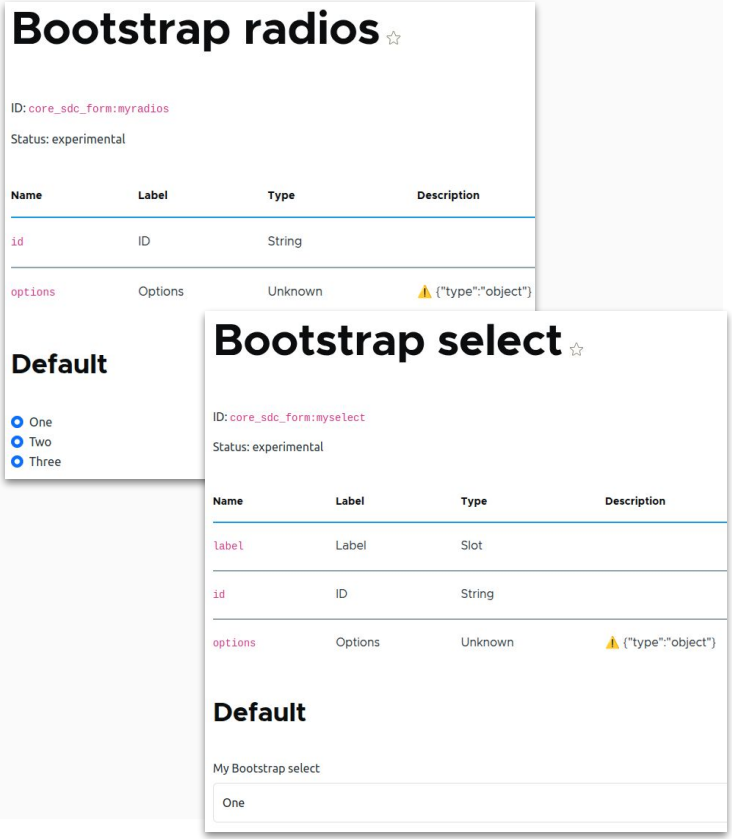
The *front devs friendly* Form API.

No PHP involved. UI focused & reusable form components.

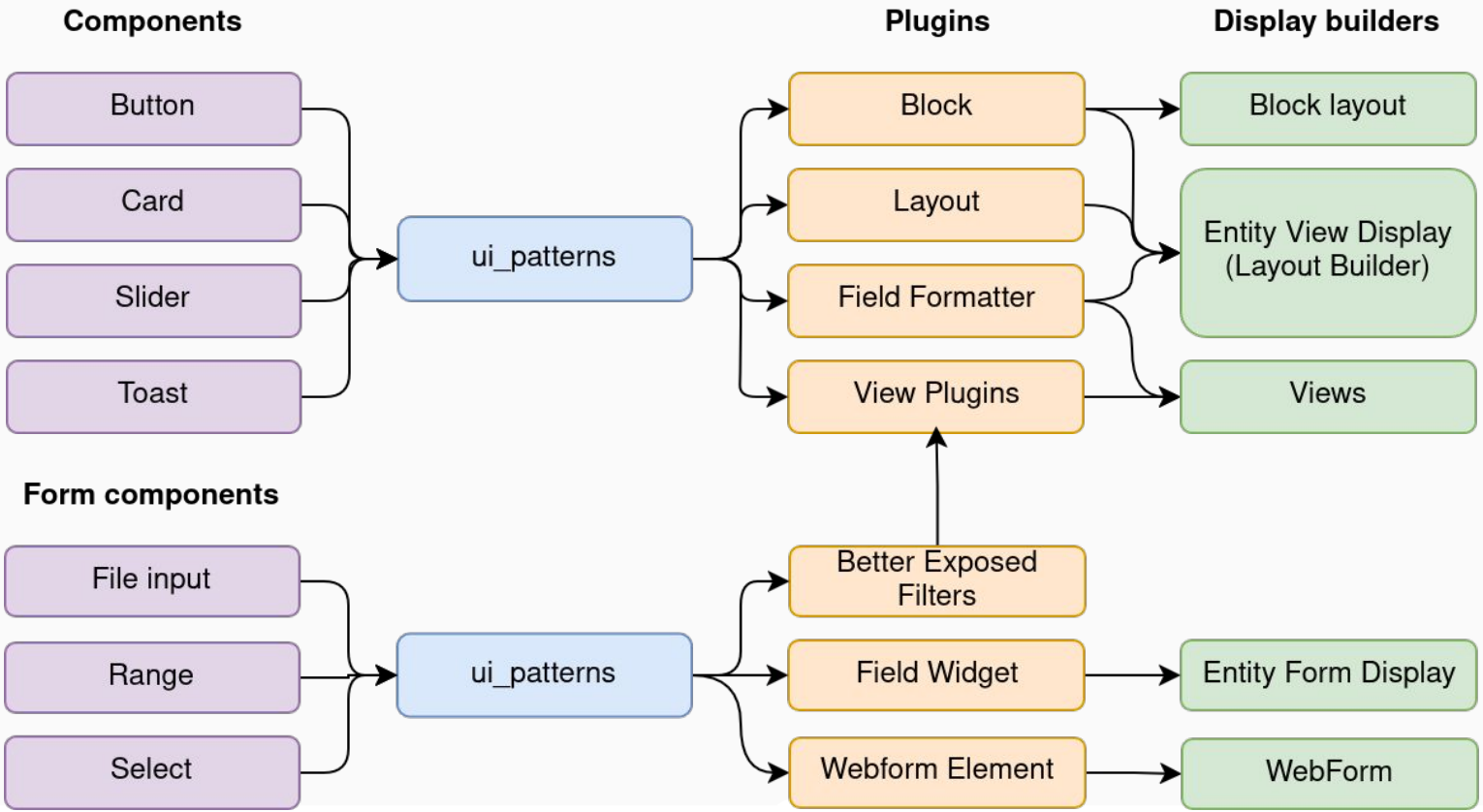
Part of the design system implementation.

Business agnostic. Sharable in a Drupal theme.

Testable out of context from a component library.



Benefit 3: Form building with SDC



Work in progress

Core issues:

- #3494634: Compatibility between SDC and the Form API
- #3508641: Define form elements from SDC

Next steps:

- Test submit button with component
- Error message handling: transmit the state into the element & message if no #title
- Handle required on checkboxes and radios (backend side) (composite elements)
- Form Ajax API & Form states API
- ComponentElement: remove #slots usage completely
- Replacing existing form elements
- Handle #tree
- …

Photo by Pixabay