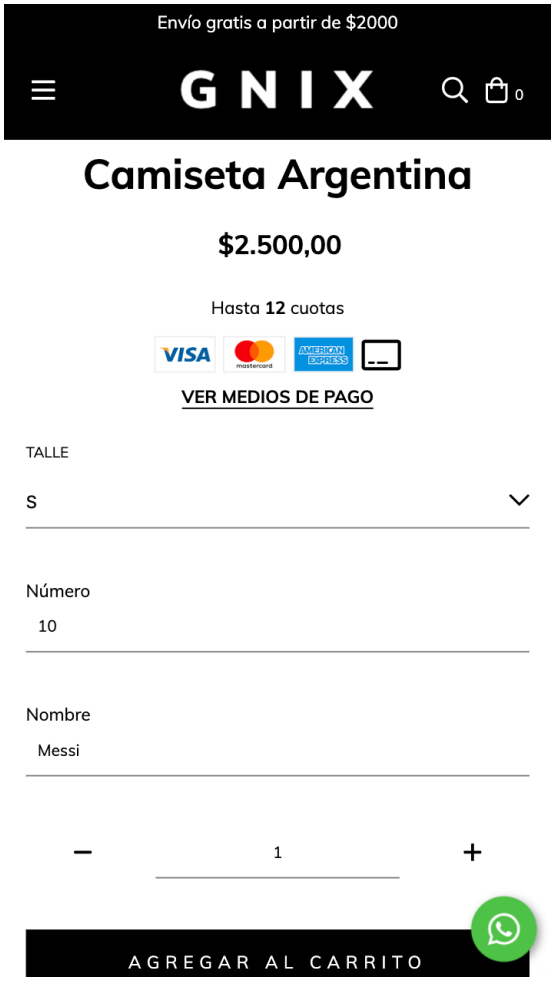
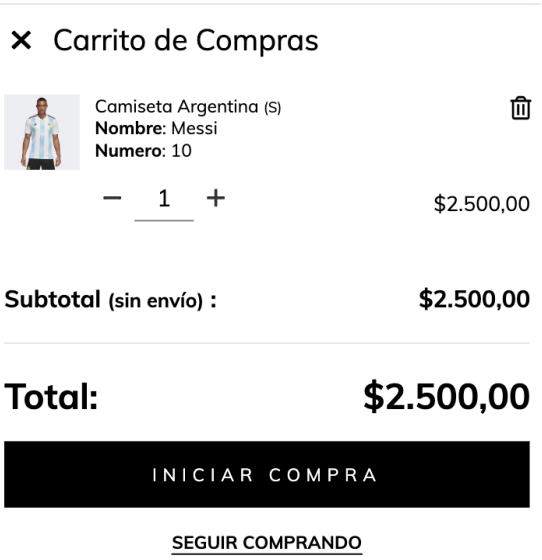
Productos personalizados

En este artículo vamos a agregar campos para ofrecer productos más personalizados. Por ejemplo para los casos donde la tienda vende remeras estampadas con un número y un nombre.

Esta información será llevada luego al checkout y por último se verá reflejada en el detalle de la orden desde el Administrador nube.





HTML

1. Creamos la carpeta forms dentro de la carpeta **snipplets**. Acá necesitamos sumar un archivo nuevo con el nombre **form-input.tpl** que vamos a usar para cada input en el detalle del producto.

```
{# /*============================================================================
========
```

```
  #Form Input
================================================================================
=====*/

#Properties

#Group
    //input_group_custom_class for custom CSS classes
#Label
    // input_label_id for ID
    // input_for for label for
    // input_label_custom_class for custom CSS classes
    // input_label_text for label text
#Prepend
    // input_prepend_content to add content before input
#Container (Only if has prepend or append)
    // form_control_container_custom_class for container custom class. E.
g: col
#Input
    // Can be text_area or input
    // input_type to define type (text, tel, number or passowrd)
    // input_id for id
    // input_name for name
    // input_value for val
    // input_placeholder for placeholder
    // input_custom_class for custom CSS classes
    // input_rows for textarea rows
    // input_data_attr for data attributes
    // input_data_val for input_data_attr value
    // input_aria_label for aria-label attribute
#Append
    // input_append_content to add content after input
#Alerts
    // input_form_alert to insert alerts
#}


    {% if input_label_text %}
        {{ input_label_text }}
    {% endif %}
    {% block input_prepend_content %}
    {% endblock input_prepend_content %}
    {% if input_append_content or input_prepend_content %}
```

```
{% endif %}
{% if text_area %}

{% else %}

{% endif %}
{% if input_append_content or input_prepend_content %}


{% endif %}
{% block input_append_content %}
{% endblock input_append_content %}
{% if input_help %}

        {% block input_help_text %}{% endblock input_help_text %}


{% endif %}
{% block input_form_alert %}
{% endblock input_form_alert %}
```

2. En el snipplet del formulario de producto, **product-form.tpl**, dentro de la carpeta **snipplets/product**; o dependiendo del diseño puede ser directamente en el template **product.tpl**, agregar un input para la propiedad que necesites. Lo importante es que esté dentro del form con el ID “product_form”.

Este input tiene que tener un atributo “name” con el valor “properties[x]”, reemplazando la “X” por el nombre de la propiedad que necesites.

Por ejemplo para un número y un nombre necesitamos algo como:

```
{% embed "snipplets/forms/form-input.tpl" with{type_text: true, input_nam
e: 'properties[Número]', input_label_text: 'Nombre' | translate } %}
{% endembed %}


{% embed "snipplets/forms/form-input.tpl" with{type_text: true, input_nam
e: 'properties[Nombre]', input_label_text: 'Nombre' | translate } %}
{% endembed %}
```

Este embed imprime un HTML similar al siguiente:

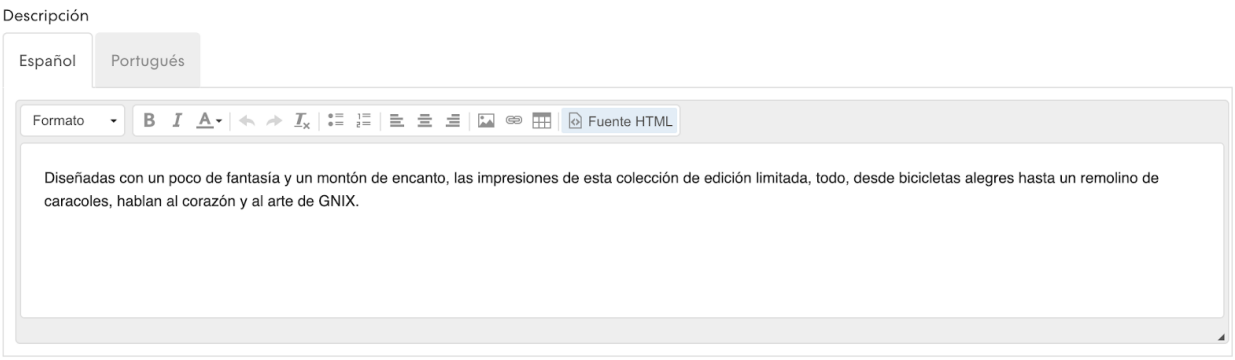
```
    {{ 'Número' | translate }}

<input type="text" name="properties[Número]" value="" />


    {{ 'Nombre' | translate }}

<input type="text" name="properties[Nombre]" value="" />
```

Si queremos que el input sea información oculta que sólo esté visible en la orden para la persona que administra la tienda, simplemente agregamos guión bajo al comienzo del nombre, por ejemplo:

```
{% embed "snipplets/forms/form-input.tpl" with{type_text: true, input_nam
e: 'properties[_Nombre]', select_label: false %}
{% endembed %}
```

Imprimiendo algo como:

Si por alguna razón necesitamos mostrar el input solo en algunos productos, entonces lo mejor es agregar estos inputs desde la descripción del producto en el Administrador nube, solo que habilitando el formato HTML.



Pero hay que considerar que este campo no usa Twig, por lo que tenemos que usar HTML “clásico”, por ejemplo:

```
{{ 'Nombre' | translate }}
```

Solo tenemos que recordar colocar el código {{ product.description }} dentro del formulario de producto ya que generalmente está por fuera del mismo.

La funcionalidad permite cualquier tipo de input (text, number, etc), solo debe estar dentro del formulario de producto para que la información sea enviada al checkout y luego a la orden generada

2. Para mostrar estas propiedades en el carrito, debemos agregar en el snipplet

cart-item-ajax.tpl, o en el template **cart.tpl** (dependiendo de tu diseño) el siguiente código:

```
{% for key, value in item.attributes if key[:1] != '_' %}

{{key}}: {{value}}

{% endfor %}
```

CSS

Requisito:

Tener agregados en tu diseño las clases helpers. Podés seguir este este pequeño tutorial para hacerlo (simplemente es copiar y pegar algunas clases, no toma más de 1 minuto).

1. Agregamos el siguiente SASS de colores en **style-colors.scss.tpl** (o la hoja de tu diseño que tenga los colores y tipografías de la tienda). Recordá que las variables de colores y tipografías pueden variar respecto a tu diseño:

```
{# This mixin adds browser prefixes to a CSS property #}


@mixin prefix($property, $value, $prefixes: ()) {
  @each $prefix in $prefixes {
      #{'-' + $prefix + '-' + $property}: $value;
  }
    #{$property}: $value;
}


{# /* // Forms */ #}


input,
textarea {
  font-family: $body-font;
}
```

```
.form-control {
  display: block;
  padding: 10px 8px;
  width: 100%;
  border: 0;
  border-bottom: 1px solid rgba($main-foreground, .5);
  -webkit-appearance: none;
  -moz-appearance: none;
  appearance: none;
  color: $main-foreground;
  background-color: $main-background;
  &:focus{
    outline: 0;
  }
  &-inline{
    display: inline;
  }
}


.form-control::-webkit-input-placeholder {
  color: $main-foreground;
}
.form-control:-moz-placeholder {
  color: $main-foreground;
}
.form-control::-moz-placeholder {
  color: $main-foreground;
}
.form-control:-ms-input-placeholder {
  color: $main-foreground;
}
```

2. Agregar los estilos dentro del archivo static/style-critical.tpl

Si en tu diseño usas una hoja de estilos para el CSS crítico, vamos a necesitar agregar el siguiente código debajo dentro de la misma, pero si no es el caso podés unificar el CSS de los pasos 2 y 3 en un solo archivo.

```
{# /* // Forms */ #}


.form-group {
  position: relative;
  width: 100%;
}
.form-group .form-select-icon{
  position: absolute;
  bottom: 12px;
  right: 0;
  pointer-events: none;
}
.form-row {
  width: auto;
  display: -webkit-box;
  display: -ms-flexbox;
  display: flex;
  -ms-flex-wrap: wrap;
  flex-wrap: wrap;
  margin-right: -5px;
  margin-left: -5px;
  clear: both;
}


.form-row > .col,
.form-row > [class*=col-]{
  padding-right: 5px;
  padding-left: 5px;
}


.form-label {
  display: block;
  font-size: 10px;
  text-transform: uppercase;
}
```

3. Agregar los estilos dentro del archivo **static/style-async.tpl**

```
{# /* // Margin and Padding */ #}


%element-margin {
  margin-bottom: 35px;
}


{# /* // Mixins */ #}


{# This mixin adds browser prefixes to a CSS property #}


@mixin prefix($property, $value, $prefixes: ()) {
  @each $prefix in $prefixes {
    #{'-' + $prefix + '-' + $property}: $value;
  }
  #{$property}: $value;
}


{# /* // Forms */ #}


.form-group{
  @extend %element-margin;
  .form-label{
    float: left;
    width: 100%;
    margin-bottom: 10px;
  }
  .alert{
    margin: 10px 0 0 0;
  }
}


{# /* Disabled controls */ #}


input,
select,
textarea{
```

```
    &[disabled],
    &[disabled]:hover,
    &[readonly],
    &[readonly]:hover{
      background-color: #DDD;
      cursor: not-allowed;
    }
  }
```

Listo, ¡Ya podés ofrecer productos personalizados en tu diseño!