INGENIERÍA | DEPARTAMENTO
UNIVERSIDAD DE LOS ANDES | DE COMPUTACIÓN
MÉRIDA VENEZUELA | ESCUELA DE SISTEMAS

Laboratorio de Ingeniería del Software
Práctica Nro. 6
Scrum y Estilos con CSS

Prof. Solazver Solé
Prep. Yenifer Ramírez
Semestre B-2018

1. Scrum

Es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos. El Scrum adopta una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto.

1.1. Roles del Equipo

- **Product Owner:** es el responsable de lo que entra en la *product backlog* y lo prioriza. El *product owner* también representa a las partes interesadas clave de un proyecto y se asegura de que los intereses del grupo estén a la vanguardia. Las personas con habilidades de comunicación de primer nivel y conocimiento del mercado suelen ser los mejores *product owners*.
- **Scrum Master:** es el facilitador del equipo. Esta persona garantiza que los equipos tengan lo que necesitan para hacer el trabajo y organiza reuniones para coordinar los esfuerzos. En lugar de poseer autoridad de gestión, estas personas permiten a sus colegas trabajar de manera más eficiente entre sí y con los departamentos relacionados.
- **Desarrolladores y Testers:** son quienes escriben el código y se aseguran de que hace lo que se supone que debe hacer. En un entorno ágil,

estos individuos son libres de elegir el método de desarrollo que consideren más adecuado para una situación. Sin sistemas estáticos prescritos, los desarrolladores tienen más espacio para innovar dentro del código.

1.2. Product Backlog

Contiene una lista de deseos con todas las historias de usuario que harán que el producto sea excelente. En Scrum, las características se conocen como *historias de usuario* y se escriben desde la perspectiva de un usuario final. El *product owner*, que representa a los usuarios y clientes de un producto, decide qué historias de usuario se convierten en el *product backlog*.

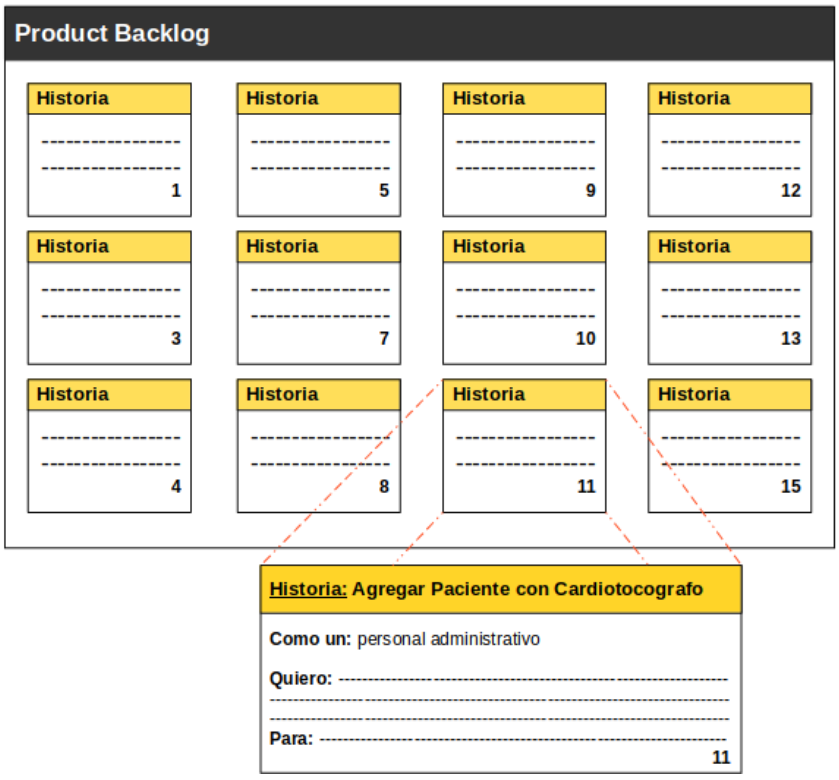

Figura 1: Ejemplo de un Product Backlog

1.3. Release Backlog

Es un subconjunto del *product backlog*, que luego son priorizadas por el equipo de desarrollo, que estiman la cantidad de tiempo necesario para completar cada elemento. Al desarrollar todos los elementos del *release backlog* se tendrá una versión del producto.

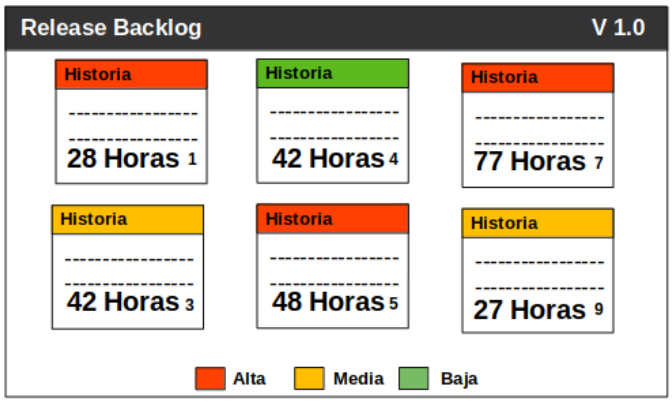

Figura 2: Ejemplo del Release Backlog

1.4. Sprint

Es un hito de corta duración, dónde se toma una parte del *release backlog* y lo lleva a una versión lista para entrega. Los sprints generalmente varían de 5 a 30 días de duración. Cuanto más corto sea el tiempo de entrega de una versión, más corto debe ser cada sprint. Al final de cada sprint, debes tener un producto completamente probado con todos los elementos en ese sprint 100 % completo.

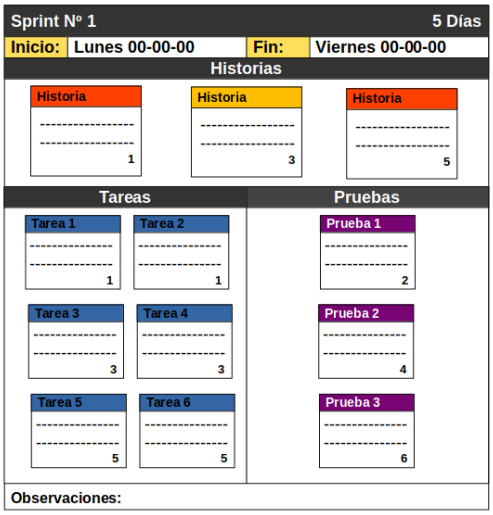

Figura 3: Tablero de un Sprint

1.5. Burndown

El progreso del equipo se rastrea mediante un gráfico de *burndown chart*, una de las mejores herramientas de visibilidad del proyecto, y representa visualmente el progreso de un proyecto. La tabla de *burndown chart* proporciona una medida diaria del trabajo que permanece en un sprint o lanzamiento dado.
La pendiente del gráfico, o velocidad de burndown chart, se calcula comparando el número de horas trabajadas con la estimación original del proyecto y muestra la tasa promedio de productividad de cada día.

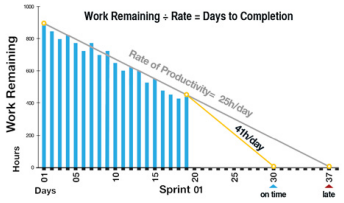

Figura 4: Burndown Chart

1.6. Daily Standups

Son periodos de trabajo cortos y diarios, también conocidos como el scrum diario. Garantizan que el sprint esté en camino y que los miembros del equipo tengan las herramientas que necesitan. Se considera una reunión que permite que la información fluya libremente y promueve la comunicación, se realiza durante el scrum, los miembros del equipo se ponen de pie en sus escritorios y enumeran el trabajo que han completado desde la última reunión. Cada persona también transmitirá cualquier obstáculo inminente e indicará qué es lo que se está preparando para hacer a continuación.

1.7. Retrospectives

Después de cada sprint, se debe hacer una reunión retrospectiva guíada por el *Scrum Master*, se utiliza en una discusión abierta para ayudar a ajustar el proceso de desarrollo para el futuro.
Este es un momento para que cada miembro del equipo reflexione sobre los aspectos del sprint que fueron bien y las áreas que podrían necesitar mejoras. A veces, el facilitador usará juegos rompehielos para facilitar al equipo a estas conversaciones perspicaces.

2. CSS

El lenguaje CSS (las siglas en inglés de hojas de estilos en cascada, o Cascading Style Sheets) sirve para describir la apariencia de un sitio web escrito en un

lenguaje de marcado (como HTML).

2.1. Bootstrap

Es un framework CSS con HTML y Javascript totalmente *responsive* que te permite dar estilos a cualquier sitio web con una configuración mínima.

2.1.1. Configurar Bootstrap en Django

Para utilizar Bootstrap debemos descargarlo en `https://getbootstrap.com/`, y obtendremos la versión compilada de CSS y JS. O podemos usar el CDN[1] de Bootstrap alojado en:

```
<link rel="stylesheet"
 href="https://stackpath.bootstrapcdn.com/bootstrap/
 4.1.3/css/bootstrap.min.css"
 integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXF
 oaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
 crossorigin="anonymous"
>

<script src="https://stackpath.bootstrapcdn.com/
bootstrap/4.1.3/js/bootstrap.min.js"
integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IM
qE241rYiqJxyMiZ6OW/JmZQ5stwEULTy"
crossorigin="anonymous">
</script>
```

2.1.2. Static Files en Django

Los archivos estáticos son los archivos CSS e imágenes. Su contenido no depende del contexto de la petición y siempre será el mismo para todos los usuarios. En Django se puede definir los archivos estáticos por aplicación así que solo se debe crear un directorio llamado `static` dentro de nuestra app (continuaremos trabajando en la aplicación `accounts` creada en la práctica anterior):

```
(eV)usuario@pc:accounts$ mkdir static
```

En este nuevo directorio vamos a crear nuestros archivos CSS y otro contenido que nos permitirán maquetar la aplicación de nuestro proyecto. Como ejemplo añadiremos un poco de estilo a nuestra página de inicio.
Para iniciar crearemos el archivo **home.css** dentro del directorio `css` creado previamente en `static`.

[1] Una red de distribución de contenidos (CDN) es una red superpuesta de computadoras que contienen copias de datos, colocados en varios puntos de una red con el fin de maximizar el ancho de banda para el acceso a los datos de clientes por la red.

```
(eV)usuario@pc:accounts$ mkdir static/css
```

```
(eV)usuario@pc:accounts$ touch static/css/home.css
```

Ahora editaremos nuestro archivo **base.html** de `templates` para añadir los CDN de Bootstrap dentro de la cabecera y el archivo **home.css**

```
1  <!--templates/base.html-->
2  {% load static %}
3  <!DOCTYPE html>
4  <html>
5  <head>
6    <meta charset="utf-8">
7    <link rel="stylesheet"
8     href="{% static 'css/home.css' %}">
9    <link rel="stylesheet"
10    href="https://stackpath.bootstrapcdn.com/bootstrap/
11    4.1.3/css/bootstrap.min.css"
12    integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXF
13    oaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
14    crossorigin="anonymous"
15   >
16   <script src="https://stackpath.bootstrapcdn.com/
17    bootstrap/4.1.3/js/bootstrap.min.js"
18    integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IM
19    qE241rYiqJxyMiZ6OW/JmZQ5stwEULTy"
20    crossorigin="anonymous">
21   </script>
22   <title>{% block title %}Django Auth Tutorial
23                  {% endblock %}</title>
24 </head>
25 <body>
26   <main>
27     {% block content %}
28     {% endblock %}
29   </main>
30 </body>
31 </html>
```

Nota: la linea *2* del código de **base.html** indica a Django que se están cargando los archivos estáticos, por lo que en la linea *15* ya sabrá de dónde se obtiene el archivo **css/home.css**. Aclaramos que el navegador lee los archivos en el orden que le son dados, por lo que debemos asegurarnos de que se llaman en el lugar correcto. De lo contrario, el código en nuestro archivo podría ser reemplazado por código de nuestros archivos Bootstrap.
Añadiremos algunas etiquetas en nuestro css (se estima que ya se entiende este código) y editamos nuestro archivo **home.html** para usarlas:

```
h1 a {
    color: #FCA205;
}
p {
    margin-bottom: 70px;
    color: #000000;
}
.page-header {
    background-color: #ff9400;
    margin-top: 0;
    padding: 20px 20px 20px 40px;
}
```

```
<!--templates/home.html-->
{% extends 'base.html' %}
<div class="page-header">
        {% block title %}<h1>Home</h1>{% endblock %}
</div>
{% block content %}
{% if user.is_authenticated %}
  Hi {{ user.username }}!
{% else %}
  <p>You are not logged in</p>
  <a href="{% url 'login' %}">login</a>
{% endif %}
{% endblock %}
```

2.1.3. Bootstrap Forms

Para explicar esta sección debemos adelantar un poco del siguiente laboratorio, por lo que vamos a agregar algunas lineas de código que serán explicadas más adelante, pero antes instalaremos lo necesario para trabajar con Crispy Forms[2]:

```
(eV)usuario@pc:~$ pip install django-crispy-forms
```

Agregamos a nuestro `INSTALLED_APPS` de **settings.py** y seleccionamos qué estilos usar

```
INSTALLED_APPS = [
    ...

    'crispy_forms',
]
```

[2]Aplicación con bootstrap que te da control sobre cómo renderizar formularios en Django

```
6
7 CRISPY_TEMPLATE_PACK = 'bootstrap4'
```

Ahora supongamos que tenemos un modelo como el siguiente, llamado `Author` y en una aplicación llamada `catalog`:

```
1  #mi_projecto/catalog/models.py
2  from django.db import models
3
4  class Author(models.Model):
5      first_name = models.CharField(max_length=100)
6      last_name = models.CharField(max_length=100)
7      date_of_birth = models.DateField(null=True,
8                                       blank=True)
9      date_of_death = models.DateField('Died',
10                                      null=True,
11                                      blank=True)
```

Además queremos crear una vista para agregar nuevos objetos a `Author`. En ese caso podríamos utilizar la vista incorporada `CreateView`:

```
1  #mi_projecto/catalog/views.py
2  from django.views.generic import CreateView
3  from .models import Author
4
5  class AuthorCreateView(CreateView):
6      model = Person
7      fields = ('first_name', 'last_name',
8                'date_of_birth', 'date_of_death')
```

Si creamos este archivo tal cual con estos código, Django intentará usar una plantilla básica de formularios llamada **author_form.html**, por lo que debemos crear este archivo y añadirle el código necesario.

```
(eV)usuario@pc:catalog$ touch templates/author_form.html
```

```
1  <!--templates/author_form.html-->
2  {% extends 'base.html' %}
3
4  {% block content %}
5    <form method="post">
6      {% csrf_token %}
7      {{ form }}
8      <button type="submit" class="btn btn-success">
```

```
            Save
    </button>
  </form>
{% endblock %}
```

Para representar el mismo formulario utilizando las clases *CSS de Bootstrap*, podemos hacer lo siguiente y mejorar los resultados:

```
<!--templates/author_form.html-->
{% extends 'base.html' %}

{% load crispy_forms_tags %}

{% block content %}
  <form method="post" novalidate>
    {% csrf_token %}
    {{ form|crispy }}
    <button type="submit" class="btn btn-success">
        Save
    </button>
  </form>
{% endblock %}
```

Hay algunos casos en los que es posible que desee más libertad para representar cada campo de nuestro formulario, y lo podemos hacer representando los campos manualmente y utilizando el filtro `as_crispy_field`:

```
<!--templates/author_form.html-->
{% extends 'base.html' %}

{% load crispy_forms_tags %}


{% block content %}
  <form method="post" novalidate>
    {% csrf_token %}
    {{ form.first_name|as_crispy_field }}
    {{ form.last_name|as_crispy_field }}
    <div class="row">
      <div class="col-6">
        {{ form.date_of_birth|as_crispy_field }}
      </div>
      <div class="col-6">
        {{ form.date_of_death|as_crispy_field }}
      </div>
```

```
    </div>
    <button type="submit" class="btn btn-success">
        Save
    </button>
  </form>
{% endblock %}
```

2.1.4. Form Helpers

La aplicación `django-crispy-forms` tiene una clase especial nombrada `FormHelper` para hacernos la vida más fácil y darnos un control completo sobre cómo deseamos representar nuestros formularios.
Para esta sección del laboratorio, haremos un ejemplo de una vista de actualización y se mostrarán los códigos de **forms.py, views.py** y **author_update_form.html**, respectivamente.
En nuestro **forms.py** el trabajo se realiza dentro del método `__init__()`. El resto es solo una forma de un modelo Django regular. Aquí se definimos que este formulario debe manejar la solicitud utilizando el método `POST` y debe tener un botón de envío con la etiqueta `"Save"`.

```
#mi_projecto/catalog/forms.py

from django import forms
from crispy_forms.helper import FormHelper
from crispy_forms.layout import Submit
from catalog.models import Author

class AuthorForm(forms.ModelForm):
    class Meta:
        model = Author
            fields = ('first_name', 'last_name',
                  'date_of_birth', 'date_of_death')

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.helper = FormHelper()
        self.helper.form_method = 'post'
        self.helper.add_input(Submit('submit', 'Save'))
```

Y por ultimo definiremos nuestra vista con su respectiva plantilla dónde llamaremos a la etiqueta nuestra instancia de formulario % crispy % como parámetro.
Y eso es todo lo que necesitamos para renderizar el formulario:

```
#mi_projecto/catalog/views.py
```

```
2
3  from django.views.generic import UpdateView
4  from catalog.models import Author
5  from catalog.forms import AuthorForm
6
7  class AuthorUpdateView(UpdateView):
8      model = Author
9      form_class = AuthorForm
10     template_name = 'catalog/author_update_form.html'
```

```
1  <!--catalog/templates/author_update_form.html-->
2  {% extends 'base.html' %}
3
4  {% load crispy_forms_tags %}
5
6  {% block content %}
7    {% crispy form %}
8  {% endblock %}
```

Recomendación: seguir el laboratorio y crear todos los archivos de la sección *CSS* y comparen los cambios obtenidos. Consulte la documentación de `https://django-crispy-forms.readthedocs.io` y aplique el estilo a los formularios de la aplicación de **autenticación** del laboratorio 5.

Referencias

[1] Scrum (desarrollo de software). (2019). Tomado de https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software).

[2] Scrum Team Roles | ScrumHub. (2019). Tomado de https://www.scrumhub.com/scrum-guide/team-roles.

[3] Scrum Product Backlog | ScrumHub. (2019). Tomado de https://www.scrumhub.com/scrum-guide/product-backlog.

[4] Scrum - Release Backlog | ScrumHub. (2019). Tomado de https://www.scrumhub.com/scrum-guide/release-backlog.

[5] Scrum Sprint - What Is A Sprint | ScrumHub. (2019). Tomado de https://www.scrumhub.com/scrum-guide/sprints.

[6] Scrum - Burndown Chart. (2019). Tomadode https://www.scrumhub.com/scrum-guide/burndowns.

[7] Daily Scrum - Daily Standup | ScrumHub. (2019). Tomado de https://www.scrumhub.com/scrum-guide/daily-standups.

[8] Scrum Retrospective - Retrospective Meeting | ScrumHub. (2019). Tomado de https://www.scrumhub.com/scrum-guide/retrospectives.

[9] more, R. (2019). How to Use Bootstrap 4 Forms With Django. Simple is Better Than Complex. Tomado de: https://simpleisbetterthancomplex.com/tutorial/2018/08/13/how-to-use-bootstrap-4-forms-with-django.html.

[10] Tutorial.djangogirls.org. (2019). CSS - ¡Que quede bonito! · Django Girls Tutorial. Tomado de: https://tutorial.djangogirls.org/es/css.