

**Proyecto Ollivander’s**

CFGS DUAL DESARROLLO DE APLICACIONES WEB – CIFP Francesc Borja Moll

Daniel Arqués Toro y Daniel Jonathan Zurita Mena

֎ Índice General

[ **Índice de Ilustraciones** 2](#_Toc29444561)

[**1.** **Introducción** 3](#_Toc29444562)

[**2.** **Entornos de Desarrollo** 4](#_Toc29444563)

[**2.1.** **Comandos Git** 4](#_Toc29444564)

[**2.2.** **Gestión del proyecto** 5](#_Toc29444565)

[**2.3.** **Clockify** 5](#_Toc29444566)

[**2.4.** **Conventional Commits** 5](#_Toc29444572)

[**1.** 6](#_Toc29444573)

[**2.5.** **Esquema UML** 6](#_Toc29444579)

[**3.** **Programación** 7](#_Toc29444580)

[**3.1.** **Flask** 7](#_Toc29444581)

[**3.2.** **Lógica de la aplicación** 7](#_Toc29444582)

[**3.2.1** **Programación orientada a objetos** 8](#_Toc29444583)

[**4.** **Lenguaje de Marcas** 9](#_Toc29444584)

[**4.1.** **HTML** 9](#_Toc29444585)

[**4.2.** **Bootstrap** 9](#_Toc29444591)

[**5.** **Base de Datos** 10](#_Toc29444592)

[**5.1.** **SQLAlchemy** 10](#_Toc29444598)

[**6.** **Sistemas Informáticos** 11](#_Toc29444599)

[**7.** **Conclusiones** 12](#_Toc29444600)

[**7.1.** **Posibles mejoras** 12](#_Toc29444601)

[**7.2.** **Dificultades** 12](#_Toc29444602)

[**8.** **Webgrafía** 13](#_Toc29444603)

# **Índice de Ilustraciones**

[Ilustración 1 - Introducción: Git Hub 3](#_Toc29444604)

[Ilustración 2 - Introducción: Repositorios y branch 3](file:///C:\Users\dajoz\Documents\FP%20DUAL\1º%20FP%20Superior%20Dual%20-%20Desarrollo%20de%20Aplicaciones%20Web\ProyectoOllivanders\Documentación_ProyectoOllivanders.docx#_Toc29444605)

[Ilustración 3 - Introducción: Presentación Word 3](#_Toc29444606)

[Ilustración 4 - Crear un repositorio en Git Hub - Comandos Git Bash 4](file:///C:\Users\dajoz\Documents\FP%20DUAL\1º%20FP%20Superior%20Dual%20-%20Desarrollo%20de%20Aplicaciones%20Web\ProyectoOllivanders\Documentación_ProyectoOllivanders.docx#_Toc29444607)

[Ilustración 5 - Conventional Commits 5](#_Toc29444608)

[Ilustración 6 - Estructura commit 5](#_Toc29444609)

[Ilustración 7 - Tipos de commits 6](file:///C:\Users\dajoz\Documents\FP%20DUAL\1º%20FP%20Superior%20Dual%20-%20Desarrollo%20de%20Aplicaciones%20Web\ProyectoOllivanders\Documentación_ProyectoOllivanders.docx#_Toc29444610)

[Ilustración 8 - Borrador UML 6](#_Toc29444611)

[Ilustración 9 – Inicialización de flask 7](file:///C:\Users\dajoz\Documents\FP%20DUAL\1º%20FP%20Superior%20Dual%20-%20Desarrollo%20de%20Aplicaciones%20Web\ProyectoOllivanders\Documentación_ProyectoOllivanders.docx#_Toc29444612)

[Ilustración 10 – Ejecutar flask en el servidor local 7](file:///C:\Users\dajoz\Documents\FP%20DUAL\1º%20FP%20Superior%20Dual%20-%20Desarrollo%20de%20Aplicaciones%20Web\ProyectoOllivanders\Documentación_ProyectoOllivanders.docx#_Toc29444613)

[Ilustración 11 - pip install flask-bootstrap 9](file:///C:\Users\dajoz\Documents\FP%20DUAL\1º%20FP%20Superior%20Dual%20-%20Desarrollo%20de%20Aplicaciones%20Web\ProyectoOllivanders\Documentación_ProyectoOllivanders.docx#_Toc29444614)

[Ilustración 12 - Bootstrap 9](#_Toc29444615)

[Ilustración 13 - templates 9](#_Toc29444616)

[Ilustración 14 – pip install flask-sqlalchemy 10](file:///C:\Users\dajoz\Documents\FP%20DUAL\1º%20FP%20Superior%20Dual%20-%20Desarrollo%20de%20Aplicaciones%20Web\ProyectoOllivanders\Documentación_ProyectoOllivanders.docx#_Toc29444617)

[Ilustración 15 - SQLAlchemy 10](file:///C:\Users\dajoz\Documents\FP%20DUAL\1º%20FP%20Superior%20Dual%20-%20Desarrollo%20de%20Aplicaciones%20Web\ProyectoOllivanders\Documentación_ProyectoOllivanders.docx#_Toc29444618)

# **Introducción**

El objetivo de este proyecto consiste en realizar una página web que guarde y actualice el stock de una tienda de objetos mágicos. Se utilizará *flask* para la creación de un entorno virtual donde estructurar la base de datos del programa y la presentación del mismo en un servidor local.

Ilustración 1 - Introducción: Git Hub

Se realiza en parejas, por lo tanto, ambos estaremos trabajando en los mismos archivos, y se encargará de ir documentando las partes del proyecto que realice.

La manera de la que trabajaremos simultáneamente en los mismos archivos será mediante *Git Hub*. El repositorio constará de ramas para el desarrollo. Cada sección del proyecto se desarrollara en una rama diferente (lógica, *templates*, bdd…). Habrá también una rama llamada *preprod* que es la primera rama de desarrollo que se usa para crear la primera parte del proyecto, que es un pequeño prototipo con la parte de lógica vinculada a una pequeña página web de ejemplo y una base de datos volátil.

La documentación del mismo tiene que estar presente con los puntos que hay en el índice de este documento.

Ilustración 2 - Introducción: Repositorios y branch

Ilustración 3 - Introducción: Presentación Word

La fecha de entrega de la primera parte del trabajo (prototipo de la lógica) será día **9 de Diciembre,** donde también tendremos que defender lo realizado hasta el momento. La fecha de entrega del resto del proyecto aún está por determinar.

El final del documento consta de una Webgrafía hypervinculada a las direcciones que se han tomado como referencia y/ o consulta para este trabajo.

# **Entornos de Desarrollo**

En este proyecto ponemos en práctica lo anteriormente aprendido sobre control de versiones, compartir repositorios, crear *branches* y hacer un *merge* entre las mismas.

## **Comandos Git**

Estos comandos son los mas utilizados en git:

* ***Git init*** : Iniciar un repositorio en local
* ***Git add .*** : Añade todos los archivos a estar *tracked* o añadidos los cambios para un próximo commit.
* ***Git commit -m “ “:*** Hace una captura de las instancias y archivos que previamente se ha hecho un *add* y que estan trackeadas. La opción m nos permite añadir un mensaje al commit en el que describimos el commit. Estos últimos dias hemos aprendido que exite un estandar de cómo se crear los comentarios.
* ***Git push -u origin master:*** Hacemos una subida del último commit al repositorio remoto donde master es la rama en la que hacemos el push. También hemos aprendido a manejar diferente ramas.

Ilustración 4 - Crear un repositorio en Git Hub - Comandos Git Bash



## **Gestión del proyecto**

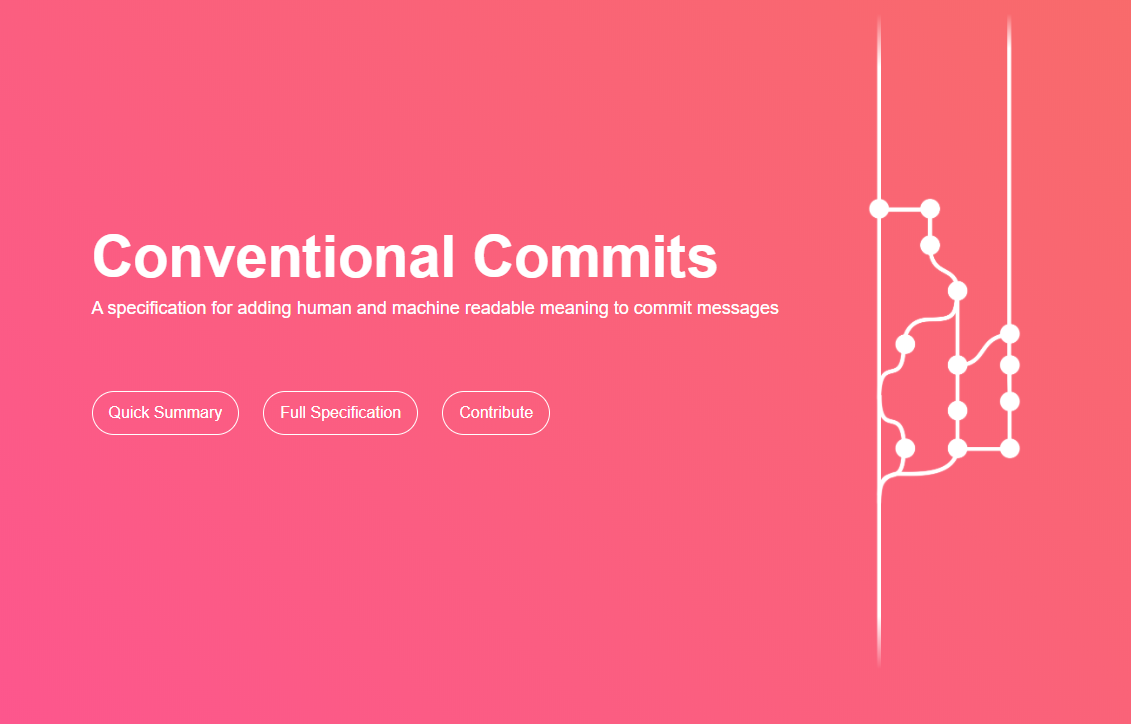
En la primera parte del proyecto se realizará el prototipo comentado anteriormente de forma colaborativa.

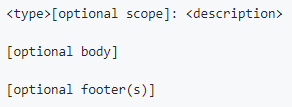
## **Clockify**



## **Conventional Commits**

Para poder hacer los *commits* de una manera que podamos entender qué tipo de modificaciones hemos hecho en cada uno utilizamos el estándar de ***Conventional Commits.***



Ilustración 5 - Conventional Commits

Este se basa en seguir una estructura donde asignamos el tipo de *commit*, a donde apuntan los cambios realizados, una breve descripción. Las partes que se encuentran entre paréntesis son opcionales y solo se añaden si el *commit* lo requiere; las que van entre “**<>**” son obligatorias.

Ilustración 6 - Estructura commit

VSCode tiene una extensión para que nos sea más fácil realizar estos *commits*.

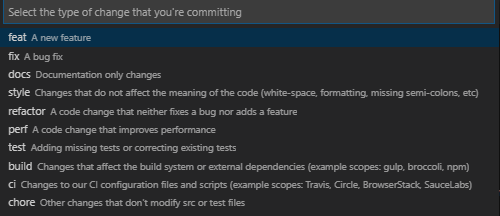


Ilustración 7 - Tipos de commits



## **Diagrama UML**

Puesto que no pudimos ver cómo crear un esquema UML, improvisamos un borrador en conjunto poniendo las ideas en común y con ayuda del profesor.

Uno de los requisitos que teníamos que cumplir es que la clase ***Item*** no podíamos modificarla de ninguna manera. Por ello, se crea una clase llamada ***NormalItem*** *que hereda todas las cualidades de* **Item** *para que podamos ir añadiéndole más funciones sin tocar la clase* **Item***. También se mencionó la clase* **Updateable** *como una interfaz que nos hace tener que sobrescribirla ya que la añadiríamos como herencia en* **NormalItem***.*

Posteriormente hay una la función que actualiza su ***quality*** y su ***sellin*** para todos los ***NormalItem*** por defecto. Luego, hay que crear una clase para cada tipo de item en la que se sobrescribe de nuevo esta función si es necesario para la lógica de cada uno.

En nuestro caso hemos implementado otras funciones no representadas en este esquema para hacer diferentes tareas.

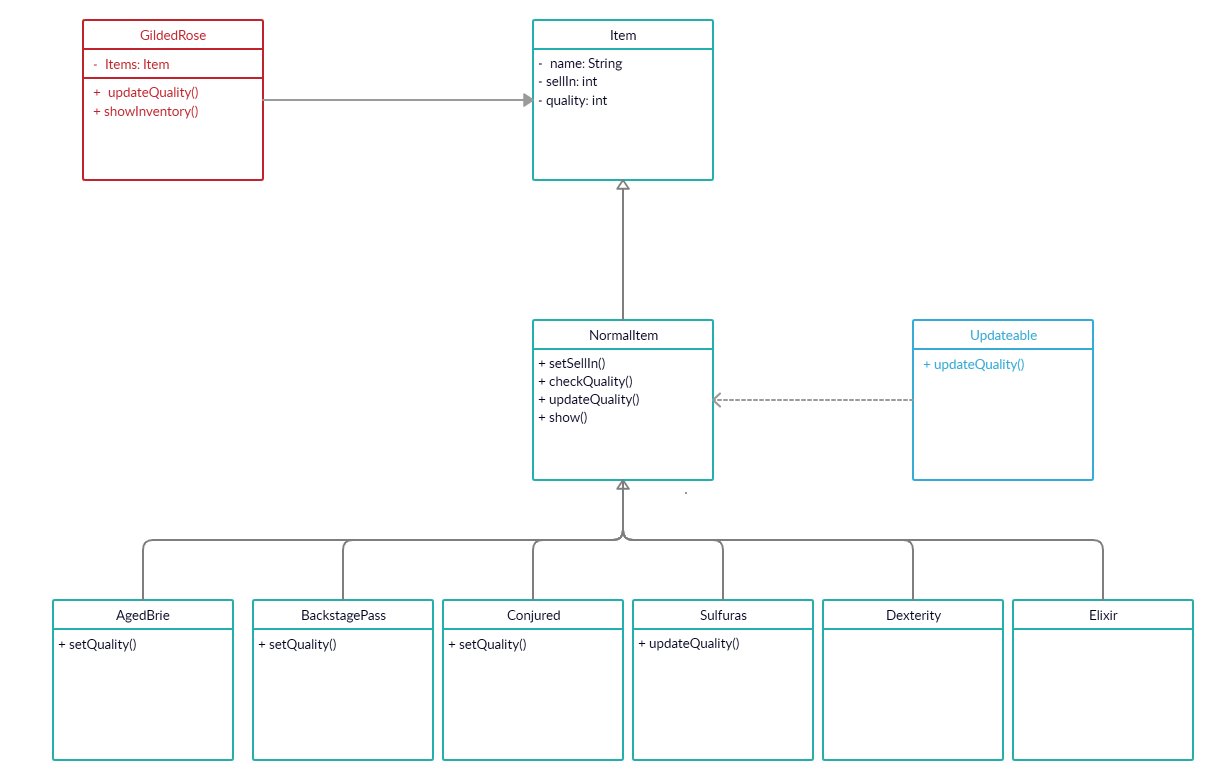


Ilustración 8 - Diagrama UML

# **Programación**

La parte de programación es la primera parte a realizar. Se crea un entorno virtual con *flask,* en el cual se crea una pequeña base de datos para los objetos en *stock* de la tienda, y se presenta el mismo en un servidor local (accesible a través de navegador).

## **Flask**

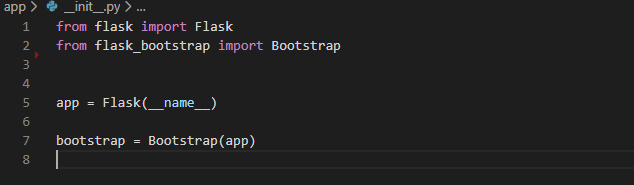
**Para crear un entorno virtual en flask, creamos dos archivos llamados \_\_init\_\_.py y routes.py. El archivo “\_\_init\_\_.py” será el constructor la app de flask. En nuestro caso también usamos un poco de Bootstrap para dejar las templates hechas más fácil.

Ilustración 9 – Inicialización de flask

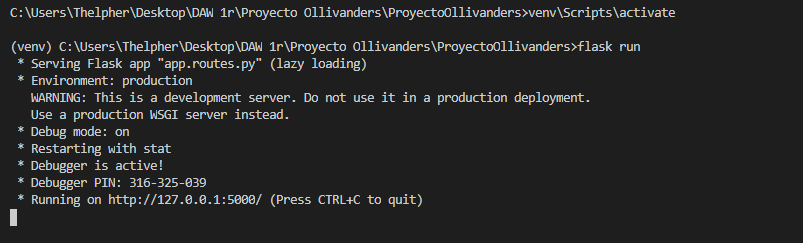
En routes.py declaramos las rutas disponibles en el servidor. Esto es particularmente útil para poder tener varias páginas en las que, por ejemplo, mostrar el stock actual en una de las rutas y en otra, permitir actualizar el inventario.

Ilustración 10 – Ejecutar flask en el servidor local

## **Lógica de la aplicación**

Cada objeto de la tienda tiene ciertas propiedades, un nombre, una fecha de venta y una calidad.

Los objetos se comportan de diferente manera, mientras que un objeto normal reduce su fecha de venta en “uno” cada día y se disminuye su calidad también en “uno” a diario, hay otros objetos que, en vez de disminuir la calidad, la aumentan, o disminuyen en otra proporción… De manera que necesitamos hacer un programa que tenga en cuenta el comportamiento de cada objeto del inventario.

## **Programación orientada a objetos**

Dado que cada objeto se comporta de manera distinta, usaremos la **POO,** es decir, haremos una estructura de objetos que heredaran propiedades de otros. De esta manera podemos tener muchas clases de *items* que hereden de una sola clase, en este caso llamada **NormalItem**.

# **Lenguaje de Marcas**

## **HTML**

Esta primera aparte del programa, consta de un sencillo HTML con tres páginas: *index.html, home.html* y *update.html.*

En *home* y en *update* es donde se muestra la ejecución del programa, pudiendo mostrar una página diferente dependiendo de la ruta que se le añada a *update (update/1* muestra la página correspondiente al primer día de *stock, update/2* la página del segundo día y así consecutivamente).



## **Bootstrap**

Usamos también las templates de boostrap para tener una plantilla que sea agradable a la vista y no solo un print de los datos. Para ello hay que instalar la correspondiente librería de flask y crear el objeto ***Bootstrap.***

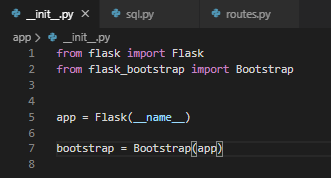


Ilustración 11 - pip install flask-bootstrap

Ilustración 12 - Bootstrap

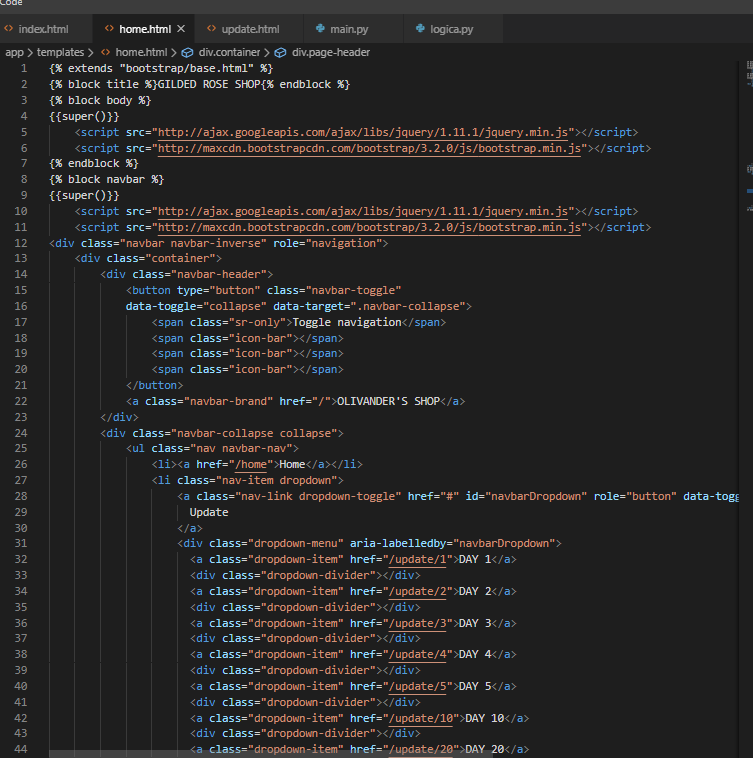
****

Ilustración 13 - templates

# **Base de Datos**

La base de datos de la primera parte del programa se realizará con el entorno Flask. La implementación de una base de datos no era obligatoria, solo para subir nota.



## **SQLAlchemy**

Primero de todo hay que instalar la correspondiente extensión de flask: ***flask-sqlalchemy*** e importar al módulo la librería ***SQLAlchemy***. También se importa la librería ***OS*** para acceder a diferentes paths necesarios para crear la base de datos.

Ilustración 14 – pip install flask-sqlalchemy

En nuestro caso hicimos la configuración de la base de datos con ayuda del libro y tenemos la plantilla de la base de datos para crear, y lista para añadir datos en ella. Lo más correcto sería añadir la clase en otro módulo para que no dependa del mismo documento en el que se crea o incluso añadir la sentencias para crear el objeto ***db*** en el \_\_init\_\_.py.

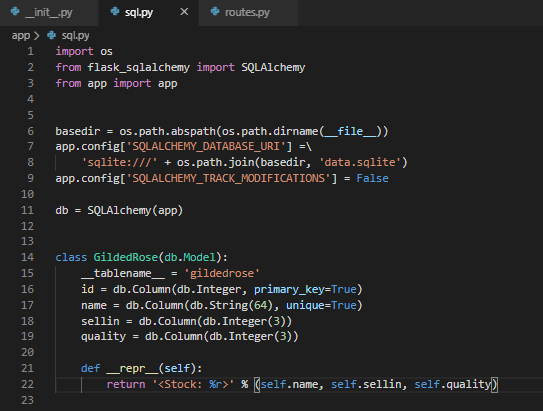
Consiste en una clase GildedRose que es la que crea la tabla con los siguientes campos: ***id, name, sellin*** y ***quality.*** También tiene una función \_\_repr\_\_ para que haga un return de los datos.

Ilustración 15 - SQLAlchemy

# **Sistemas Informáticos**

La asignatura de sistemas informáticos no ha tenido prácticamente nada de repercusión en el proyecto.

La única mención sería a la hora de crear las carpetas de dentro del proyecto para tenerlo todo ordenado y el uso de la librería ***OS*** para crear la base de datos con flask.

# **Conclusiones**

## **Posibles mejoras**

* Durante navidades y fin de año no se trabajó nada en el proyecto.
* El uso de la función \_\_repr\_\_ en todo el proyecto ha sido prácticamente nula.
* Los casos test solo reflejan el comportamiento de todos los objetos a la vez.
* Podríamos hacer sido tan recurrentes con los commits como en el proyecto anterior.
* No ha habido pair programming como dios manda.

## **Dificultades**

* Entender la programación orientada a objetos por nuestra cuenta.
* El uso de flask por nuestra cuenta.
* Printar el return de las funciones que teníamos con flask.
* Por motivos que desconocemos flask no consigue importar modulos en otros pcs.
* Fortuito push -f

# **Webgrafía**

* *Clockify*
  + [*https://clockify.me/tracker*](https://clockify.me/tracker)
* *Documentación Flask* 
  + [*http://flask.palletsprojects.com/en/1.1.x/*](http://flask.palletsprojects.com/en/1.1.x/)