



PROYECTO OLIVANDERS

Proyecto transversal

DESCRIPCIÓN BREVE

Proyecto transversal Olivanders; Lógica, Flask y Base de datos.

Alberto Mañas y Mateu Massó

1r DAW DUAL

Índice

Definición general del proyecto de software.....	2
Procedimientos de desarrollo.	2
Herramientas utilizadas:.....	2
Planificación:	2
Procedimientos de instalación y prueba.	3
Requisitos no funcionales:	3
Obtención e instalación:.....	3
Definición general del proyecto de software:.....	4
Arquitectura del sistema.....	4
Diagrama UML:.....	5
Descripción individual de los módulos:.....	5
Gilded_rose.py	5
Item.py	5
Updatable.py	6
Normal_item.py	6
Aged_brie.py	6
Backstage_pass.py	6
Conjured.py	6
Sulfuras_hand.py.....	7

Índice ilustraciones

Il·lustració 1: Diagrama en sucio hecho en clase	3
Il·lustració 2: Gilded_rose.py.....	4
Il·lustració 3: Diagrama UML.....	5

Definición general del proyecto de software

En este proyecto consiste en la creación de una página web desarrollada con python y con un framework de este llamado Flask. Tras el desarrollo de la parte lógica, procederemos a enlazar dicha lógica con la aplicación creada a partir de Flask, esto nos permitirá poder generar en el puerto 5000 lo desarrollado.

Procedimientos de desarrollo.

Herramientas utilizadas:

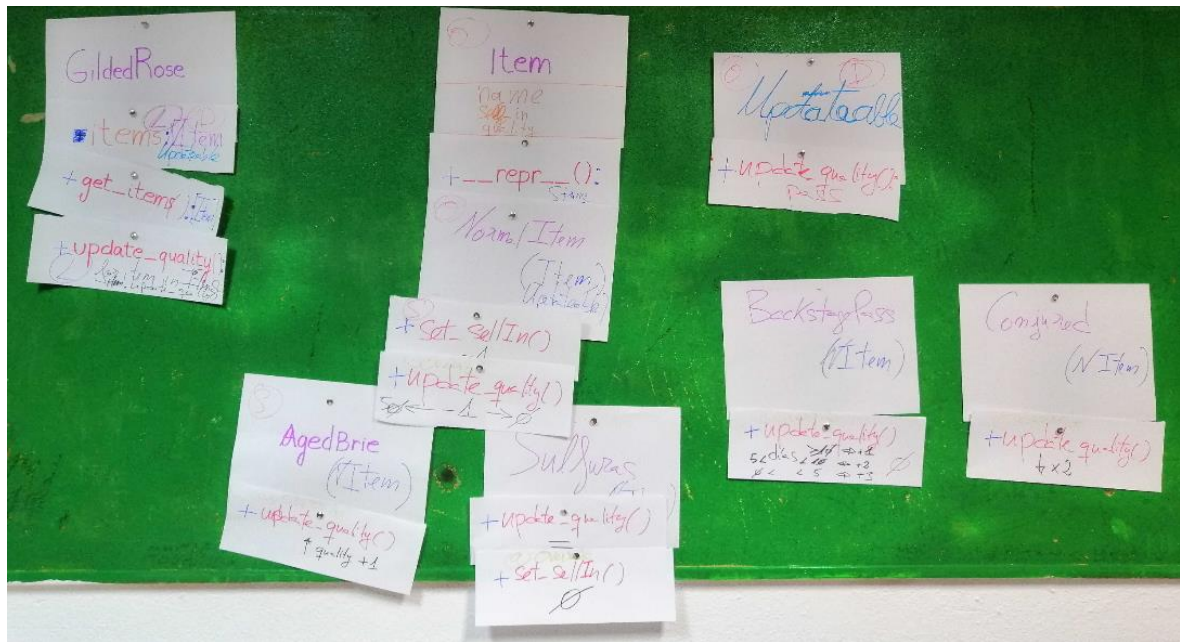
Nuestro IDE de desarrollo ha sido VS code, dónde hemos hecho la lógica y aplicado Flask, un Framework de python que contiene las herramientas necesarias para desarrollar una página web.

Además, hemos llevado a cabo un desarrollo teniendo en cuenta el control de versiones utilizando git con varias ramas para desarrollar y realizar las pruebas pertinentes con éxito.

Planificación:

La planificación de este proyecto fue resuelta en una Daily con todo el equipo de desarrollo y con el product owner. Tras haber hecho un analisis en completo de como se desarrollaría dicho proyecto.

Para ello, hemos utilizado TDD para todos los módulos utilizando una metodología en cascada, la cual consiste en desarrollar el código acabando módulo por módulo antes de continuar con el siguiente. Esto ha sido más sencillo que la primera vez por haber tenido ya anteriormente un primer contacto con esta metodología.



Il·lustració 1: Diagrama en sucio hecho en clase

Procedimientos de intalación y prueba.

Requisitos no funcionales:

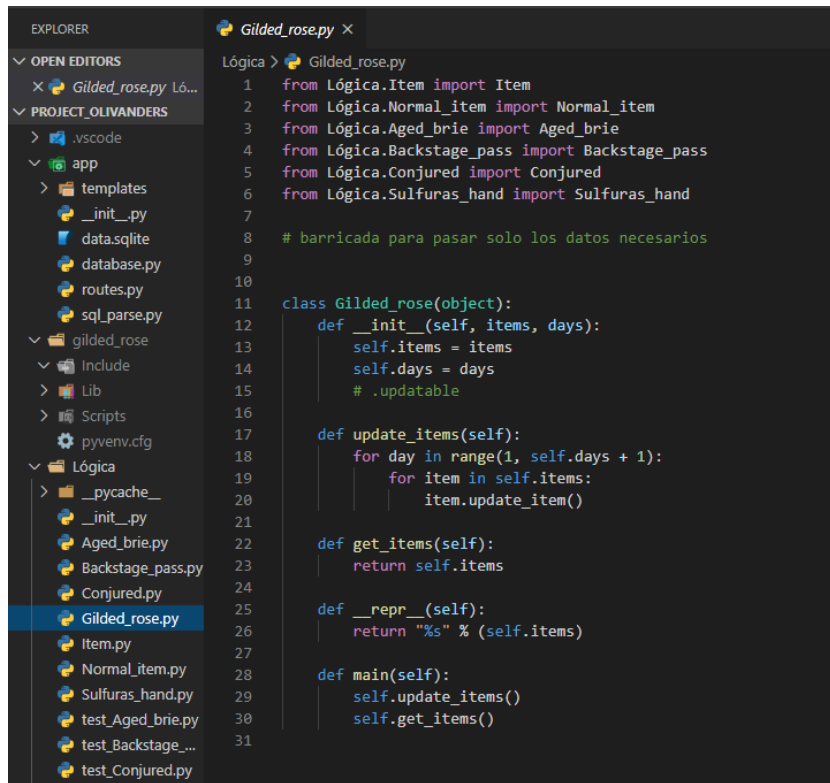
Durante el desarrollo han surgido varios problemas que los hemos ido solucionando, pero en especial el hecho de obtener datos de la base de datos y manipularlos hecha por SQLAlchemy de Flask, hasta el día de hoy seguimos desarrollando esta parte.

Obtención e instalación:

El programa debe ser ejecutado teniendo python y en un IDE compatible con éste. Por otra parte, debemos crear el entorno virtual de la aplicación utilizando Flask, para ello es recomendable exportar en un fichero .txt las dependencias utilizadas para que los integrantes del proyecto coincidan con estas y poder realizar el trabajo con éxito.

Es importante destacar que, en cuanto se desee trabajar con este entorno, activarlo utilizando un script llamado `activate` con path: `venv\Scripts\activate`. Éste proceso es modificado levemente dependiendo de que SO se utilice por el programador.

Tras tener lo anterior completado y el entorno listo para el funcionamiento, debemos proceder a abrir el archivo `Lógica\Gilded_rose.py`, el cual hará de main para este proyecto.



The screenshot shows a code editor with a dark theme. On the left, the 'EXPLORER' sidebar displays the project structure for 'PROJECT_OLIVANDERS'. The 'Lógica' folder is expanded, showing files like '_pycache_', '_init_.py', 'Aged_brie.py', 'Backstage_pass.py', 'Conjured.py', 'Gilded_rose.py' (which is selected), 'Item.py', 'Normal_item.py', 'Sulfuras_hand.py', and test files. The main editor window shows the code for 'Gilded_rose.py'. It starts with imports from 'Lógica' for 'Item', 'Normal_item', 'Aged_brie', 'Backstage_pass', 'Conjured', and 'Sulfuras_hand'. A comment indicates a 'barricada para pasar solo los datos necesarios'. The 'Gilded_rose' class is defined with an '__init__' method, an 'update_items' method that iterates through items and updates their quality, a 'get_items' method, a '__repr__' method, and a 'main' method that calls 'update_items' and 'get_items'.

```
1 from Lógica.Item import Item
2 from Lógica.Normal_item import Normal_item
3 from Lógica.Aged_brie import Aged_brie
4 from Lógica.Backstage_pass import Backstage_pass
5 from Lógica.Conjured import Conjured
6 from Lógica.Sulfuras_hand import Sulfuras_hand
7
8 # barricada para pasar solo los datos necesarios
9
10
11 class Gilded_rose(object):
12     def __init__(self, items, days):
13         self.items = items
14         self.days = days
15         # .updatable
16
17     def update_items(self):
18         for day in range(1, self.days + 1):
19             for item in self.items:
20                 item.update_item()
21
22     def get_items(self):
23         return self.items
24
25     def __repr__(self):
26         return "%s" % (self.items)
27
28     def main(self):
29         self.update_items()
30         self.get_items()
31
```

Il·lustració 2: Gilded_rose.py

Definición general del proyecto de software:

La funcionalidad del proyecto es listar el inventario de la tienda y actualizar la calidad de cada item dependiendo de los días que se deseen actualizar.

Para ello hay distintos tipos de items hechos en distintas clases que son heredadas directamente de Item cómo por ejemplo Normal_item, cuya clase servirá de plantilla para gran parte de los objetos requeridos en este programa. Tendrá ciertas características estáticas que serán aplicadas al resto de items, cómo por ejemplo el límite fijado en quality de $0 \geq 50$.

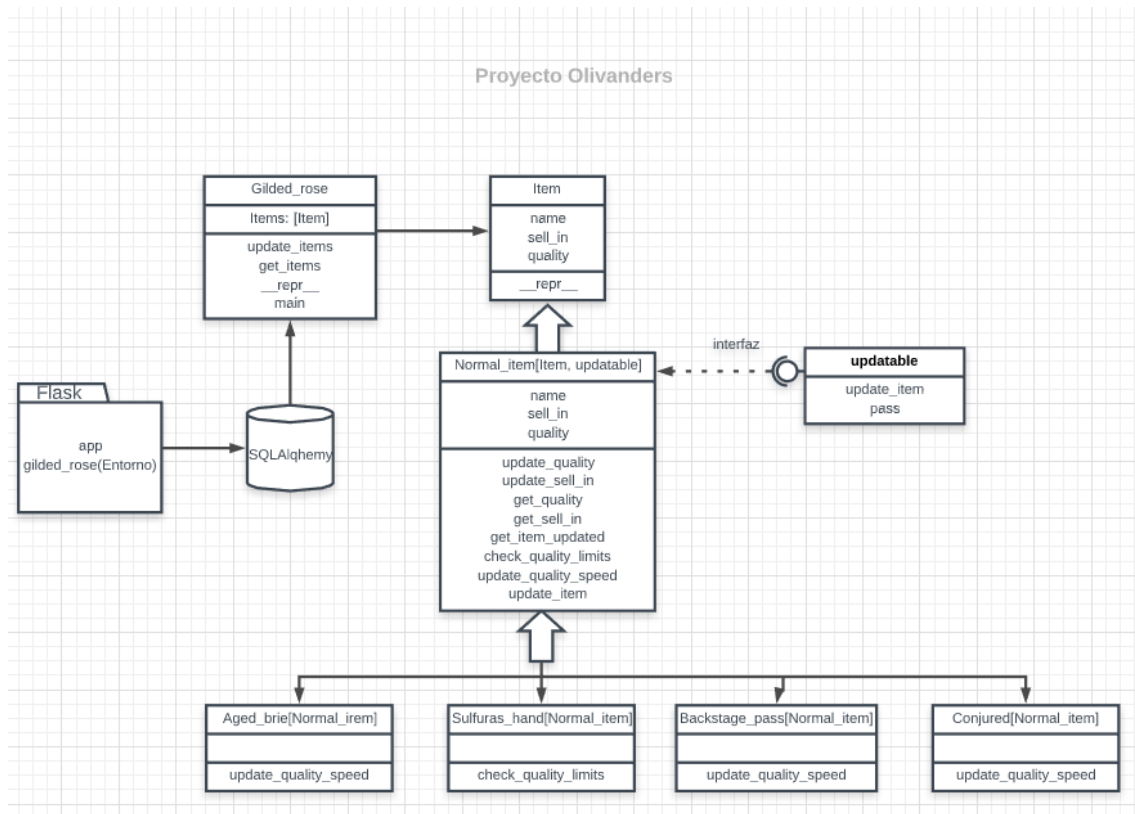
La lógica irá actualizando la calidad dependiendo de si se cumplen los requisitos fijados en ella.

Arquitectura del sistema.

La arquitectura del sistema es el corazón del programa, es la forma en la que están estructurados los módulos que contiene dicho programa. En nuestro caso hemos aplicado una arquitectura modular donde Gilded_rose hará la función de main y llamará a los otros módulos para que se ejecuten con éxito.

Además, utilizando el Framework de python llamado Flask, hacemos una pequeña base de datos de una tabla que utilizando otro módulo podremos parsear y obtener los datos con mayor éxito.

Diagrama UML:



Il·lustració 3: Diagrama UML

Descripción individual de los módulos:

En este apartado se explicará detalladamente la función de cada módulo a detalle, sus dependencias e implementación de cada uno.

Gilded_rose.py

DESCRIPCIÓN GENERAL Y PROPÓSITO → Llama a todos los otros módulos para poder ejecutar el programa con efectividad, actualizar objetos y obtenerlos.

URL REPOSITORIO

https://github.com/mmasso/project_olivanders/blob/master/L%C3%B3gica/Gilded_rose.py

Item.py

DESCRIPCIÓN GENERAL Y PROPÓSITO → Define **name**, **sell_in** y **quality** que será herencia directa a **Normal_item**.

URL REPOSITORIO

https://github.com/mmasso/project_olivanders/blob/master/L%C3%B3gica/Item.py

Updatable.py

DESCRIPCIÓN GENERAL Y PROPÓSITO → Definir update_item, lo usaremos de conector

URL REPOSITORIO →

https://github.com/mmasso/project_olivanders/blob/master/L%C3%B3gica/updatable.py

Normal_item.py

DESCRIPCIÓN GENERAL Y PROPÓSITO → Aplica name, sell_in y quality heredados de Item, además de, proporcionar las propiedades básicas de quality y sell_in, sus condiciones (límites) y por último actualizar los items.

Se aplica el concepto de quality_speed y sell_in_speed para introducir cómo se manifestará por norma la actualización de estos valores.

URL REPOSITORIO →

https://github.com/mmasso/project_olivanders/blob/master/L%C3%B3gica/Normal_item.py

Aged_brie.py

DESCRIPCIÓN GENERAL Y PROPÓSITO → Coje como referencia Normal_item pero su manifestación de quality_speed es de 1 y el sell_in se conserva igual que normal item, al igual que sus límites.

URL REPOSITORIO →

https://github.com/mmasso/project_olivanders/blob/master/L%C3%B3gica/Aged_brie.py

Backstage_pass.py

DESCRIPCIÓN GENERAL Y PROPÓSITO → Coje como referencia Normal_item y en este caso su quality_speed se incrementa 1, sell_in sigue la regla de Normal_item.

URL REPOSITORIO →

https://github.com/mmasso/project_olivanders/blob/master/L%C3%B3gica/test_Backstage_pass.py

Conjured.py

DESCRIPCIÓN GENERAL Y PROPÓSITO → Coje como referencia Normal_item, perderá por debajo de 0 el doble que normal item y continua con la misma regla de sell_in que Normal_item.

URL REPOSITORIO →

https://github.com/mmasso/project_olivanders/blob/master/L%C3%B3gica/Conjured.py

Sulfuras_hand.py

DESCRIPCIÓN GENERAL Y PROPÓSITO → Coje como referencia Normal_item, en este caso éste item no manifiesta ningún cambio de quality ni de sell_in pero el límite fijado se fija a 80

URL REPOSITORIO →

https://github.com/mmasso/project_olivanders/blob/master/L%C3%B3gica/Sulfuras_hand.py

Planificación y duración

Computo clockify:

https://github.com/mmasso/project_olivanders/blob/master/L%C3%B3gica/updatable.py

Dificultades :

Nuestra mayor dificultad en este proyecto el poco conocimiento que teníamos en un inicio de programación orientada a objetos. Esto supuesto que los integrantes hayamos tenido que buscar información exhaustivamente en las documentaciones ofrecidas.