



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática



TFG del Grado en Ingeniería Informática

Framework web de uso de
sistemas de machine learning.
Documentación Técnica



Presentado por Javier Martínez Riberas
en Universidad de Burgos — 18 de septiembre
de 2017

Tutores: Dr. José Francisco Díez Pastor
Dr. César Ignacio García Osorio

Copyright (C) 2017 Javier Martínez Riberas. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included as a pdf file titled “LICENSE.pdf”.

Índice general

Índice general	I
Índice de figuras	III
Índice de tablas	IV
Apéndice A Plan de Proyecto Software	1
A.1. Introducción	1
A.2. Planificación temporal	1
A.3. Estudio de viabilidad	4
Apéndice B Especificación de Requisitos	8
B.1. Introducción	8
B.2. Objetivos generales	8
B.3. Catálogo de requisitos	8
B.4. Especificación de requisitos	9
Apéndice C Especificación de diseño	15
C.1. Introducción	15
C.2. Diseño de datos	15
C.3. Diseño arquitectónico	16
C.4. Diseño procedimental	18
C.5. Diseño de paquetes	19
Apéndice D Documentación técnica de instalación	23
D.1. Introducción	23
D.2. Manual de instalación para programadores	23
D.3. Instalación para uso local	26
Apéndice E Documentación técnica de programación	28

E.1. Introducción	28
E.2. Estructura de directorios	28
E.3. Manual del programador	30
E.4. Despliegue	42
E.5. Pruebas del sistema	42
E.6. Configuración	42
E.7. Subproducto: Seshat	44
Apéndice F Documentación de usuario	46
F.1. Introducción	46
F.2. Requisitos de usuarios	46
F.3. Instalación	46
F.4. Manual del usuario	46
Bibliografía	52

Índice de figuras

B.1. Diagrama de casos de uso	14
C.1. Diagrama entidad relación	16
C.2. Model view controller	17
C.3. Arquitectura de la aplicación	18
C.4. Diagrama de secuencia	19
D.1. Clonación de los repositorios	24
D.2. Creación de entorno virtual e instalación de dependencias	25
D.3. Prueba de uso de pip freeze	25
E.1. Proceso de traducción	37
F.1. Punto de entrada a la aplicación	47
F.2. Pantalla de registro de usuario	47
F.3. Pantalla de inicio de sesión	48
F.4. Inicios de sesión alternativos	48
F.5. Cambio de las zonas accesibles al iniciar sesión	48
F.6. Pantalla de acceso al servicio de clasificación	49
F.7. Pantalla de subida del dataset	49
F.8. Pantalla de reentrenamiento del modelo	50
F.9. Pantalla del servicio de clasificación tras el reentrenado	50
F.10. Pantalla de clasificación tras reentrenado del modelo	51

Índice de tablas

A.1. Costes de personal	4
A.2. Costes de material al mes	5
A.3. Dependencias del proyecto en produccion	6
A.4. Dependencias exclusivas del desarrollo	7
B.1. CU-01 Registro de usuario.	10
B.2. CU-02 Login.	10
B.3. CU-03 Logout.	11
B.4. CU-04 Clasificación de imagen.	11
B.5. CU-05 Subida de datasets.	12
B.6. CU-06 Re-entrenado de modelos.	13

Plan de Proyecto Software

A.1. Introducción

La planificación es un punto importante en cualquier proyecto. Estimar el trabajo, el tiempo y el dinero que va a suponer la realización del proyecto, aunque vaya a cambiar más tarde, es interesante para saber si puede haber posibilidades de que sea viable. Para ello, debemos analizar cuidadosamente los componentes del proyecto. Con este análisis pretendemos conocer los requisitos del proyecto y pretendemos que mediante modificaciones siga sirviendo en un futuro.

A.2. Planificación temporal

En un principio se planteó seguir una metodología ágil, esta sería *Scrum* [10] ya que existía experiencia anterior. Por supuesto, no se pudo usar completamente ya que no se tenía un equipo, no se hicieron reuniones diarias. . .

Se empezó a usar ZenHub como tablero kanban donde se situarían las tareas con sus costes. Este tipo de planificación con *milestones* e *issues* no ha sido seguido la mayor parte del tiempo, ya que no era la metodología más adecuada. Las labores de investigación que no tenían ningún entregable claro no se incluyeron en el repositorio a partir de cierto punto para no acumular demasiada información innecesaria.

En retrospectiva, se ha concluido que una forma mejor de hacer este tipo de trabajos es llevar varios tableros kanban en varios repositorios: uno para todo el proyecto y otro para todo el informe del proyecto en vez de todo junto.

Sistema de *sprints*

El sistema de *sprints* no fue un constante a lo largo del desarrollo del proyecto, esto se debió al hecho de que algunas cosas se hicieran antes de empezar

el cuatrimestre por cuestiones de balanceamiento de la carga de trabajo.

Lo que esto supuso es que, antes de empezar con el sistema de *sprints*, se usase el tiempo de 1-3 *sprints* para aprender Flask, tras este periodo, se llevó un sistema más estandarizado de *sprints*, estos se pueden ver en los *milestones* de GitHub y en las *issues*. Por problemas en cómo se movieron las *issues* por el tablero (a QA hasta la reunión del *sprint*), no se pudieron sacar gráficos automatizados suficientemente exactos, ya que acababan siendo un peldaño enorme en la reunión del *sprint*.

Tras una serie de semanas llevando este sistema, se tubo que abandonar por varios motivos. Los *sprints* empezaron a ser menos concretos y más basados en la investigación y desarrollo de la alternativa más atrayente, la carga de trabajo del resto de asignaturas se tenía en picos, propiciando *sprints* con mucho trabajo y otros sin casi nada y, por último, una serie de circunstancias personales hicieron que acabase por abandonar la documentación de cada *sprint online*.

Sprints antes del cuatrimestre

Antes del cuatrimestre como se ha comentado se siguieron una serie de *sprints*, estos se basaron en seguir el tutorial *explore Flask*. El *sprint* se llevo a cabo con éxito, terminando aquello que se tenía pensado hacer, una aplicación Flask.

Sprint 1

En este *sprint* se planteó el hecho de mover el trabajo anterior a una plataforma como Git y llevar el desarrollo más formalmente. También se empezó a investigar sobre subida de archivos y subida de archivos tipo *drag and drop*. La subida de archivos se consiguió con un poco de retraso, pero el *drag and drop* no. Esto se debe a mi inexperiencia en desarrollo *front end*.

Sprint 2

En este *sprint* se empezó a subir el código a Heroku, comprobando que se podía desplegar en condiciones reales. Se intentó mejorar la interfaz gráfica.

Sprint 3

Se empezó a generar documentación fuera de código. Este *sprint* fue lento y no muy productivo, ya que no se sabía cómo se quería desarrollar la documentación. En este punto se desarrolló como una comparativa entre todas las alternativas hasta el momento, lo cual no ha sido la dirección que ha tomado el proyecto con el tiempo.

Sprint 4-6

En esta serie de *sprints* se planteó la elección del *framework* de *Machine Learning* que usar. Se investigaron tres alternativas [sklearn](http://scikit-learn.org/stable/)¹, [Theano](http://www.deeplearning.net/software/theano/)² y [Tensorflow](https://www.tensorflow.org/)³, aunque de tener la opción ahora se hubiese elegido [Keras](https://keras.io/)⁴.

Keras parece una muy buena opción debido a que tiene una *API* de alto nivel y muy intuitiva, si se quiere ver un ejemplo el siguiente [script de Keras](https://github.com/fchollet/keras/blob/master/examples/mnist_cnn.py)⁵ quizá sea un buen ejemplo de lo sencillo que puede ser el usar Keras. Además proporciona estabilidad en la *API* y permite usar Tensorflow, Theano o [CNTK](https://docs.microsoft.com/en-us/cognitive-toolkit/)⁶ (*Microsoft Cognitive Toolkit*) *backends*.

También se empezó a implementar la integración, algo que no se pudo cumplir y que acabó resultando mucho más costoso de lo que se había pensado previamente.

Sprint 7

En este *sprint* se pretendía mejorar la aplicación y documentar mucho de los *sprints* anteriores. En este punto se empiezan a tener problemas con el seguimiento de las tareas de manera online.

Docker

Tras investigar sobre el tema, se decidió poner algo de énfasis en el despliegue, ya que ya se había intentado y llevado a cabo con *Heroku*. De los contenedores existentes se tubo que investigar y decir si usarlos y cuál. La decisión fue *Docker*, al ser la única alternativa que tenía proyectos en producción. En el momento de la elección era totalmente *open source*, algo que cambió con el tiempo y tras lo cuál hubo que hacer otra investigación de cómo influía el cambio. Resultó no influir apenas, puede ser que una vez maduren otras alternativas haya que intentar deshacerse de *Docker*.^[9]

Transfer learning

Tras investigar un poco sobre cómo se podía hacer la integración con Tensorflow, se vió que el *transfer learning* de un modelo como inception v3 era suficientemente simple como para intentar llevarlo a cabo. Se desarrollaron los *scripts* para predecir clases a partir de imágenes.

¹[sklearn](http://scikit-learn.org/stable/): <http://scikit-learn.org/stable/>.

²[Theano](http://www.deeplearning.net/software/theano/): <http://www.deeplearning.net/software/theano/>.

³[Tensorflow](https://www.tensorflow.org/): <https://www.tensorflow.org/>.

⁴[Keras](https://keras.io/): <https://keras.io/>.

⁵script de Keras: https://github.com/fchollet/keras/blob/master/examples/mnist_cnn.py.

⁶[CNTK](https://docs.microsoft.com/en-us/cognitive-toolkit/): <https://docs.microsoft.com/en-us/cognitive-toolkit/>.

Docker Compose y Swarm

Docker Compose y *Docker Swarm* fueron una de las razones por las que se eligió *Docker* en vez de otras alternativas, pero aún así la barrera de entrada a estas tecnologías estaba lo suficientemente alta como para que *Swarm* acabase siendo inalcanzable. [5]

Microservicios

Los microservicios se empezaron a conocer gracias a *Docker*, ya que son una alternativa muy usada en este tipo de sistemas, al tener una relación simbiótica. Cada uno beneficia al otro.

Escalabilidad y despliegue de páginas web

Por último, se decidió centrar el proyecto en parte en la escalabilidad, ya que en dos de los ejes ya estaba, y para implementarlo en el tercero sólo hacía falta cambiar de una imagen de la base de datos a otra. Este último cambio no se llevo a cabo porque, salvo en aplicaciones excesivamente grandes, el usar una base de datos normal es más beneficioso.

A.3. Estudio de viabilidad

En este apartado se estudiarán los costes en los que se incurren al desarrollar este proyecto.

Viabilidad económica

El proyecto incurre en distintos tipos de costes

Costes de personal

El proyecto se lleva a cabo por un desarrollador junior empleado a tiempo parcial (30h/semana) durante cuatro meses. Se considera el siguiente salario:

Concepto	Coste (€)
Salario neto	1000
Retención IRPF (19 %)	360.53
Seguridad social (28,30 %)	537.00
Salario bruto	1897.53
4 meses tiempo parcial(3/4)	5692.59

Tabla A.1: Costes de personal

Costes de material: hardware y software

Como material podemos considerar lo mínimo necesario para llevar un proyecto así:

Un único coste puntual (ordenador portátil) que aproximamos en 600€ y en la tabla se pondrá su coste amortizado contando con una amortización a 4 años. Se ha comprobado que internet está incluido en el alquiler de la oficina.

Costes totales

El sumatorio de todos los costes es de 6188,59€. Podríamos recortar más en ciertos puntos, pero debido a que no se va a llevar a cabo como está planteado aquí, esto solo es una aproximación del coste de oportunidad.

Beneficios

Si nuestro interés fuese vender el proyecto, este no sería el proyecto que venderíamos, tendríamos que añadir medidas de tiempo computacional en cada operación, de manera que cada clasificación o re-entrenamiento cobrásemos cierta cantidad de dinero en relación al tiempo computacional usado. Esto se traduce a una explotación del tipo SaaS (Software as a Service).

De esta manera podemos crear planes para cada usuario, podemos plantearnos hacer un plan gratuito y varios planes de pago según cantidad de tiempo computacional que se le permita usar al cliente.

Viabilidad legal

La licencia necesaria para nuestro proyecto, debido a las dependencias que tiene, tendrá que ser compatible con aquellas de las bibliotecas que hemos usado. La licencia más restrictiva que hemos usado es la de paramiko siendo LGPL (2.1 y 3.0)[3], esta licencia está pensada para librerías e incluye el siguiente párrafo:

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library".

Concepto	Coste (€)
Ordenador portátil	25
Alquiler de oficina	99
1 mes	124
4 meses	496

Tabla A.2: Costes de material al mes

Such a work, in isolation, is not a derivative work of the Library,
and therefore falls outside the scope of this License.

Por lo que las restricciones de esa licencia no se nos aplican. Esto quiere decir que podemos publicar nuestro código bajo la licencia que mejor nos parezca o incluso podríamos mantenerlo privado como un Secreto de negocio (*trade secret*) ya que proporcionamos SaaS. Basándome en las licencias *open source* más comunes, ya que me parece interesante el hecho de que otras personas puedan usar el código, y con ayuda de las recomendaciones de GNU [1], se ha decidido usar la GNU GPL 3.0.

Para la documentación se ha decidido que la mejor licencia para las necesidades era la GNU FDL (GNU Free Documentation License). La segunda alternativa más interesante era Creative Commons.

Para completar la documentación sobre la viabilidad legal se adjunta una tabla con todas las dependencias del proyecto.

Dependencia	Versión	Licencia
Alembic	0.9.5	MIT
Bcrypt	3.1.3	Apache 2.0
Flask	0.12.2	BSD
Flask-Babel	0.11.2	BSD
Flask-Bcrypt	0.7.1	BSD
Flask-Login	0.4.0	MIT
Flask-SQLAlchemy	2.2	BSD
Flask-Oauthlib	0.9.4	BSD
Flask-WTF	0.14.2	BSD
itsdangerous	0.24	BSD
Jinja2	2.9.6	BSD
Paramiko	2.2.1	LGPL-3.0
Psycpg2	2.7.3	LGPL-3.0
SQLAlchemy	1.1.13	MIT
Werkzeug	0.12.2	BSD
WTForms	2.1	BSD

Tabla A.3: Dependencias del proyecto en produccion

Dependencia	Versión	Licencia
codeclimate-test-reporter	0.2.3	MIT
coverage	4.4.1	Apache 2.0
Pytest	3.2.1	MIT

Tabla A.4: Dependencias exclusivas del desarrollo

Especificación de Requisitos

B.1. Introducción

Este apartado consistirá en una descripción de los propósitos y requerimientos de la aplicación.

B.2. Objetivos generales

El objetivo principal del proyecto es el crear una aplicación web, esta debe tener control de usuarios y capacidad de clasificar imágenes mediante una red neuronal. Por último, debe ser capaz de re-entrenar el modelo de clasificación de imágenes.

La red neuronal a usar es Inception v3 [11] y el conjunto de datos es Imagenet [6].

Además se pide que todo esto se haga de una manera escalable, desplegable y con un enfoque que reduzca la deuda técnica.

B.3. Catálogo de requisitos

Requisitos funcionales

- **RF-1 Control de usuarios:** la aplicación tiene que tener capacidades para controlar usuarios.
 - **RF-1.1 Registro de usuarios:** el usuario debe poder registrarse en la aplicación con su correo.
 - **RF-1.2 Login:** el usuario debe poder entrar a la aplicación una vez esté registrado.
 - **RF-1.3 Logout:** el usuario debe poder finalizar una sesión.
 - **RF-1.4 Integración con Google:** en caso de que el usuario tenga cuenta de Google, podrá usarla para hacer el login.

- **RF-2 Uso de aprendizaje automático:** se debe permitir el uso de un sistema de aprendizaje automático basado en una red neuronal.
 - **RF-2.1 Clasificación de imágenes:** el usuario podrá clasificar imágenes individuales.
 - **RF-2.2 Re-entrenamiento del modelo:** se deberá permitir el re-entrenamiento de la red neuronal con distintos conjuntos de datos.
- **RF-3 Manejo de imágenes:** el usuario podrá subir imágenes para ser usadas por los algoritmos de aprendizaje automático.
 - **RF-3.1 Subir imagen individual:** se necesita poder subir imágenes individuales para su posterior clasificación.
 - **RF-3.2 Subir conjunto de imágenes:** se podrá subir un conjunto de imágenes con un formato determinado para poder re-entrenar los modelos.

Requisitos no funcionales

- **RNF-1 Usabilidad:** la aplicación debe ser intuitiva y con una interfaz sencilla.
- **RNF-2 Responsividad:** la interfaz tiene que cambiar de tamaño y dimensiones con respecto a la pantalla y su tamaño.
- **RNF-3 Escalabilidad:** la aplicación debe poder aumentar su rendimiento con el incremento de recursos hardware.
- **RNF-4 Seguridad:** Los datos sensibles, como las contraseñas, deben tratarse de forma segura. También se tendrá que tener las medidas básicas de seguridad contra los ataques a web más comunes.
- **RNF-5 Mantenibilidad:** la aplicación debe ser desarrollada de manera que permita escalabilidad a la hora de añadir características.
- **RNF-6 Internacionalización:** la aplicación deberá estar preparada para soportar varios idiomas.
- **RNF-7 Facilidad de despliegue:** se debe poder desplegar la aplicación en poco tiempo.

B.4. Especificación de requisitos

En esta sección se desarrollarán los casos de uso de la aplicación, cabe destacar que los actores del sistema corresponden a la misma persona en distintos momentos de su uso de la aplicación.

CU-01	Registro de usuario
Autor	Javier Martínez Riberas
Requisitos asociados	RF-1.1

CU-01	Registro de usuario
Descripción	Permite al usuario registrarse.
Precondición	Ninguna.
Acciones	<ol style="list-style-type: none"> 1. El usuario rellena el formulario de registro (email y contraseña). 2. Si está configurada la confirmación por email, se necesita confirmar para terminar el registro.
Postcondición	El usuario se registra en la base de datos.
Excepciones	<ul style="list-style-type: none"> ■ Fallo de la configuración de email. (Si no hay, no se da)
Importancia	Alta

Tabla B.1: CU-01 Registro de usuario.

CU-02	Login
Autor	Javier Martínez Riberas
Requisitos asociados	RF-1.2, RF-1.4
Descripción	Permite al usuario autenticarse.
Precondición	El usuario debe de estar registrado.
Acciones	<ol style="list-style-type: none"> 1. El usuario rellena el formulario de login (email y contraseña). O el equivalente en Google.
Postcondición	El usuario consigue una sesión.
Excepciones	<ul style="list-style-type: none"> ■ Fallo de rellenado del formulario.
Importancia	Alta

Tabla B.2: CU-02 Login.

CU-03	Logout
Autor	Javier Martínez Riberas
Requisitos asociados	RF-1.3
Descripción	El usuario se des-autentica.
Precondición	El usuario debe de estar autenticado.
Acciones	<ol style="list-style-type: none"> 1. El usuario hace click el botón de logout.
Postcondición	El usuario cierra una sesión, la cual se elimina.
Excepciones	Ninguna.
Importancia	Alta

Tabla B.3: CU-03 Logout.

CU-04	Clasificar imagen
Autor	Javier Martínez Riberas
Requisitos asociados	RF-2.1, RF-3.1
Descripción	Permite al usuario obtener la clase de una imagen.
Precondición	El usuario debe de estar autenticado.
Acciones	<ol style="list-style-type: none"> 1. El usuario sube una imagen. 2. Opcional. El usuario elige un re-entrenamiento específico.
Postcondición	El usuario ve la clase más probable y otras dos probables.
Excepciones	<ul style="list-style-type: none"> ■ Fallo de formato de imagen.
Importancia	Alta

Tabla B.4: CU-04 Clasificación de imagen.

CU-05	Subida de datasets
Autor	Javier Martínez Riberas

CU-05	Subida de datasets
Requisitos asociados	RF-3.2
Descripción	Permite al usuario subir un conjunto de imágenes para re-entrenar la red neuronal.
Precondición	El usuario debe de estar autenticado.
Acciones	<ol style="list-style-type: none"> 1. El usuario elige un dataset en su ordenador para subir. 2. El dataset debe tener una estructura de carpetas equivalentes a las clases. 3. Cada carpeta equivalente a una clase debe contener imágenes en jpg de la clase en cuestión.
Postcondición	El usuario tras un periodo de tiempo tiene disponible un dataset para re-entrenar el modelo.
Excepciones	<ul style="list-style-type: none"> ■ Fallo a la hora de subir el zip.
Importancia	Media-Alta

Tabla B.5: CU-05 Subida de datasets.

CU-06	Re-entrenado de modelos
Autor	Javier Martínez Riberas
Requisitos asociados	RF-2.2, RF-3.2
Descripción	Permite al usuario re-entrenar la red neuronal.
Precondición	El usuario debe de estar autenticado y haber subido un dataset.
Acciones	<ol style="list-style-type: none"> 1. El usuario elige un dataset. 2. Tras el tiempo requerido por los algoritmos de re-entrenamiento, el usuario tendrá disponible un modelo re-entrenado.
Postcondición	El usuario tras un periodo de tiempo tiene disponible un modelo.

CU-06	Re-entrenado de modelos
Excepciones	<ul style="list-style-type: none">Fallo del formato de las imágenes o de la estructura de archivos.
Importancia	Media-Alta

Tabla B.6: CU-06 Re-entrenado de modelos.

Por último, se expone el diagrama de casos de uso como resumen de la especificación de requisitos.

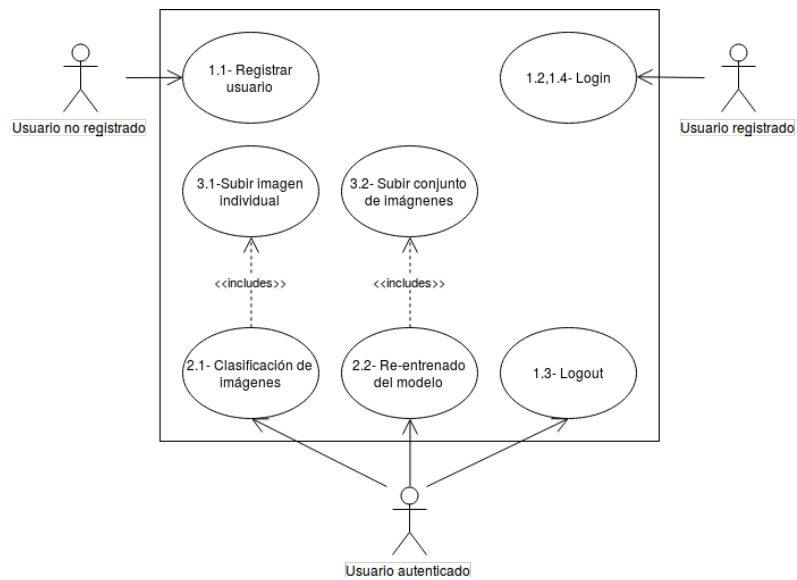


Figura B.1: Diagrama de casos de uso

Especificación de diseño

C.1. Introducción

En este anexo se expone el diseño que se ha usado para llevar a cabo los objetivos anteriores. Como se manejan los datos, la arquitectura y modelos.

C.2. Diseño de datos

La aplicación cuenta con el modelado de los siguientes elementos:

- **Usuario:** la entidad del usuario es una entidad simple la cual tiene un id auto-generado para poder identificar a cualquier usuario. También dispone de email, contraseña y confirmación del email. Por último, dispone de un campo para posibles *tokens* de *oauth* de manera que se puedan añadir distintas maneras de registrar usuarios sin tener que cambiar constantemente el modelo de la base de datos.
- **Red neuronal:** la red neuronal solo tiene lo que podríamos considerar *primary key* en una base de datos y el objeto de bytes. Su identificador se compone del nombre de usuario y del nombre de conjunto de datos que se ha usado para entrenarlo.
- **Conjunto de datos:** el conjunto de datos se identifica mediante su nombre y del usuario que lo ha subido. Tiene un objeto de bytes asociado que guardamos debido a la lentitud del entrenamiento de una red neuronal y si este entrenamiento se interrumpe, no queremos repetir la subida del conjunto de datos al servidor.

Dentro de esta configuración tenemos que tener en cuenta que sólo el modelo del usuario está dentro de la base de datos de manera que el resto se controla por software y se guarda dentro de los volúmenes de *docker*

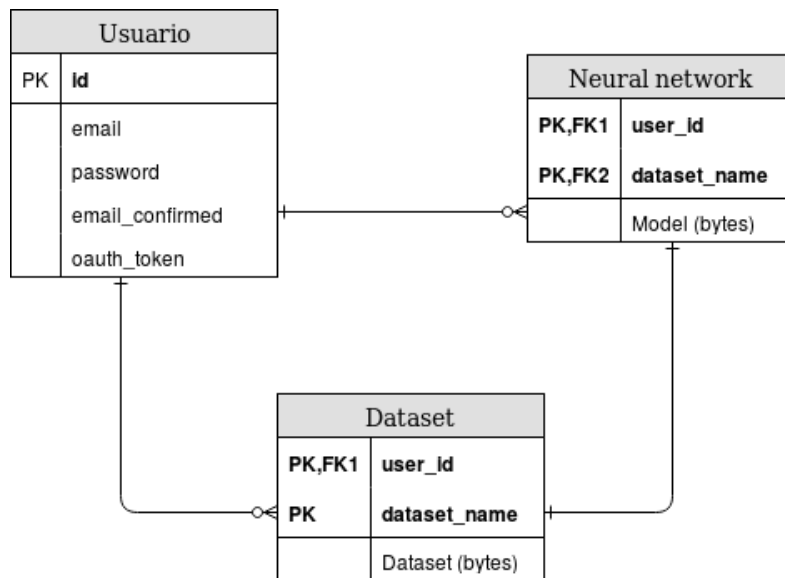


Figura C.1: Diagrama entidad relación

Paso de datos

Para proporcionar mayor seguridad el paso de datos entre los servicios se hace a través de SSH. Otra opción es tener micro servidores de Flask para exponer una *API Rest* que nos permita hacer las operaciones de una forma más elegante. Esto dificulta algo la configuración interna poniendo más peso en el equipo de *devops*, pero nos permite tener un sistema mucho más desacoplado lo cual facilita el mantenimiento de ambas partes por separado. Si se quisiera tener seguridad tras ese cambio, las comunicaciones se deberían hacer con SSL.

C.3. Diseño arquitectónico

La decisión de proporcionar el servicio como una página web ha condicionado la arquitectura. Al buscar la escalabilidad dentro del diseño se ha tenido que tener en cuenta para el diseño final. A continuación se exponen las partes más conocidas del diseño y el resultado final.

Model View Controller (MVC)

La parte de la aplicación tiene algo más de diseño en su interior. Para facilitar el desarrollo de futuros desarrolladores se ha decidido seguir la arquitectura ampliamente conocida como Modelo Vista Controlador o MVC.

De esta manera logramos separar la vista con la cual interactuará el usuario, con la lógica de interacción con la aplicación y del modelo de nuestro

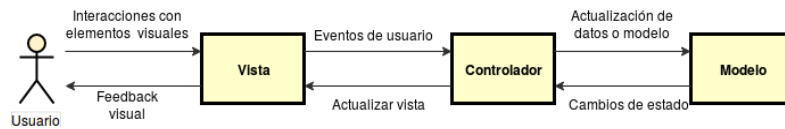


Figura C.2: Model view controller

negocio. Gracias a esta abstracción podemos ocultar toda la complejidad de cada capa a los desarrolladores de las otras dos en caso de que nuestro negocio crezca hasta ese punto.

En nuestra estructura de archivos las vistas están bajo `WhatAClass/WhatAClass/templates`, los controladores se encuentran en `WhatAClass/WhatAClass/blueprints` y el único modelo explícito es el de usuario, situado en `WhatAClass/WhatAClass/models`, el resto son estructuras lógicas sobre el sistema de archivos.

Builder

En la aplicación que hemos creado se ha usado un builder para permitir la creación de varias instancias de la aplicación de manera sencilla, ya que la aplicación es un agregado de varios compuestos. Además permite que se pueda escalar verticalmente (añadiendo más recursos en la misma máquina).

Esto también facilita la creación de la aplicación para programadores nuevos que desconozcan nuestro proyecto. Esto se debe a que los componentes están bien establecidos, lo cual permite añadir nuevos componentes o quitar otros de manera fácil.

Iterator

Aunque no se ha implementado explícitamente Python permite la iteración sobre elementos de una colección de manera transparente. Esto podría no considerarse como un patrón de diseño ya que está incorporado en el lenguaje.

Null Object

El objeto de comunicación con el servidor de correo electrónico responde como un Null Object cuando esta funcionalidad no está habilitada. Esto permite un cambio de comportamiento de manera simple sin condicionales excesivas en sitios inadecuados para tratar con el cambio en el comportamiento al estar habilitada la funcionalidad.

Arquitectura final

El diseño arquitectónico final es uno muy parecido a los que se usan en muchas páginas web para permitir mayor escalabilidad y picos de servicio sin

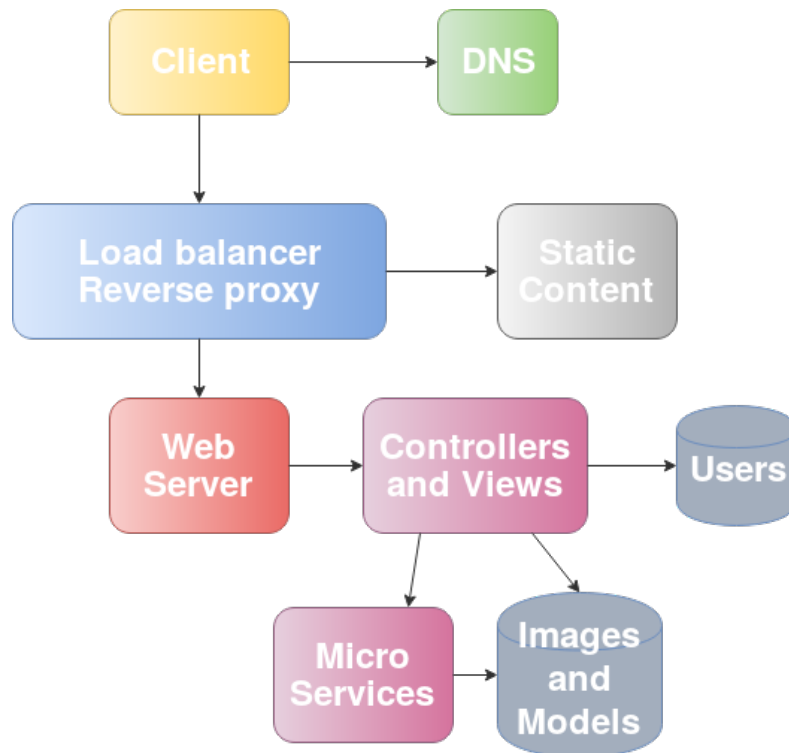


Figura C.3: Arquitectura de la aplicación

caída del mismo. Simplemente se basa en permitir ejecutar múltiples instancias de los mismos objetos. Se podría ampliar la escalabilidad mediante *reverse proxies* en distintos puntos de la arquitectura y con un servicio de caché como *Redis*¹ antes de las bases de datos para permitir mayor velocidad de acceso. Si quisiésemos replicar microservicios, deberíamos poner un *load balancer* o unirlos desde el que ya está.

C.4. Diseño procedimental

En este apartado se expondrá que flujo sigue la información de forma general cuando se usa la aplicación.

Como se puede observar la secuencia pasa por nginx para poder servir archivos estáticos a gran velocidad y para poder, si se quisiese, balancear la carga entre varios servidores de aplicación (uWSGI).

¹ *Redis*: <https://redis.io/>.

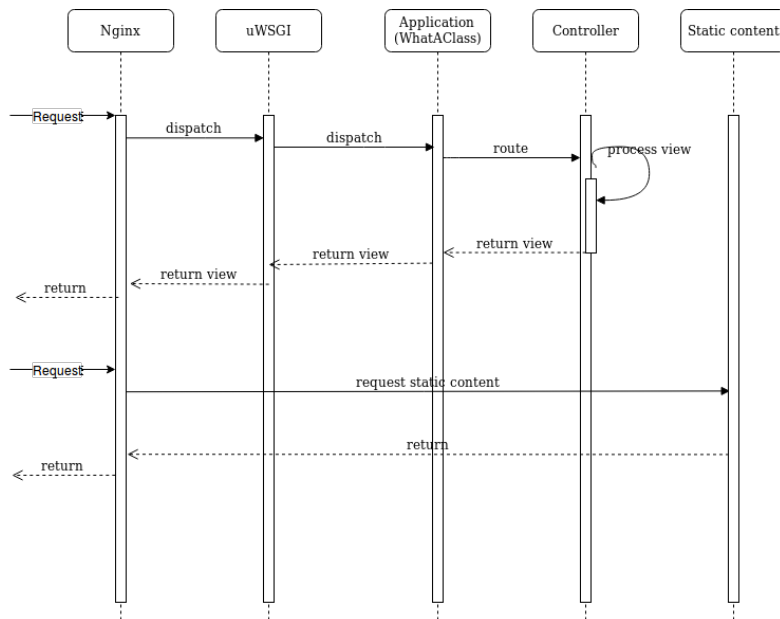


Figura C.4: Diagrama de secuencia

C.5. Diseño de paquetes

El diseño de la organización de las diferentes partes de la aplicación en paquetes, tradicionalmente, se puede hacer de dos maneras: descomposición funcional o divisional.

A continuación se pone un ejemplo de cada descomposición.

Descomposición funcional

```

yourapp/
  __init__.py
  static/
  templates/
    home/
    control_panel/
  views/
    __init__.py
    home.py
    control_panel.py
  models.py
  
```

Descomposición divisional

```
yourapp/  
  __init__.py  
  home/  
    __init__.py  
    views.py  
    static/  
    templates/  
  control_panel/  
    __init__.py  
    views.py  
    static/  
    templates/  
  models.py
```

Descomposición de la aplicación

La descomposición que se ha elegido en el proyecto es la funcional.

Como se puede ver, Python no necesita una clase para guardar código, esto resulta en un diagrama de clases más simple que en otros lenguajes como Java.

```
WhatAClass/  
  blueprints/  
    oauth/  
      __init__.py  
      google.py  
    __init__.py  
    index_blue.py  
    tensorflow_mng_blue.py  
    user_mng_blue.py  
  static/  
    css/  
    fonts/  
    js/  
    favicon.ico  
  templates/  
    tensorflow_mng/  
      predict.html  
      predicted.html  
      predicted-custom.html  
      retrain.html
```

```
    select-predict.html
    upload_ds.html
user_mng/
    email/
        activate.html
        recover.html
    login.html
    other_logins.html
    recover.html
    reset.html
    signup.html
error.html
index.html
js_upload.html
layout.html
macros.html
translations/
utils/
    __init__.py
    email.py
    redirect.py
__init__.py
app.py
controllers.py
extensions.py
forms.py
models.py
```

Aplicación

Como se puede ver la aplicación está descompuesta en el builder (“app.py”), extensiones, formularios (“templates”), modelo, utilidades y controladores.

Los controladores están en la carpeta blueprints, pero se incluyen dentro de “controllers.py” para reducir el acoplamiento.

Las utilidades tienen una configuración parecida a los controladores, las utilidades están en el “utils” y se importan con “util.py”.

Configuración

En la jerarquía de carpetas superior al código de la aplicación está la carpeta “config”, usada por el patrón builder para generar la aplicación.

En este momento solo se dispone de una configuración (default) esto se debe a que gracias al uso de *docker* y *docker compose* podemos cambiar la

configuración mediante las variables de entorno que estos pueden modificar, esto simplifica las configuraciones situacionales, ya que de esta manera toda la configuración que debamos cambiar podemos cambiarla en el archivo “docker-compose.yml”.

Tensorflow

En el microservicio de tensorflow no necesitamos una estructura compleja ya que la biblioteca es suficientemente completa como para poder usarla con scripts sencillos.

Documentación técnica de instalación

D.1. Introducción

En esta sección se procederá a explicar cómo instalar el servicio tanto para seguir desarrollando o para su explotación.

D.2. Manual de instalación para programadores

En la siguiente sección se expondrá como se puede continuar desarrollando la aplicación. Desde la preparación del entorno de desarrollo pasando por la obtención del código fuente del proyecto hasta la ejecución y despliegue del proyecto y sus test.

Entorno de desarrollo

Los siguientes elementos son necesarios para continuar el desarrollo:

- Python 3.5 o 3.6 [2]
- Git [13]
- Gestor de paquetes [7]
- Entorno virtual [8]
- IDE o editor de texto
- Docker [12]

Las instalaciones que necesitan de una instalación específica con relación al sistema operativo como pueden ser Python, Git, Docker y la IDE (Pycharm o similares) no se explicará.

```

Terminal - jack@jack-ISM: ~/dir/tfg/WhatAClass
File Edit View Terminal Tabs Help
-rw-r--r-- 1 jack jack 14 dic 12 2016 .xscreensaver
-rw----- 1 jack jack 82 ago 19 03:25 .xsession-errors
-rw----- 1 jack jack 272 ago 18 20:20 .xsession-errors.old
jack@jack-ISM:~$ cd algo
bash: cd: algo: No such file or directory
jack@jack-ISM:~$ mkdir dir
jack@jack-ISM:~$ cd dir
jack@jack-ISM:~/dir$ git clone https://github.com/Jazriel/tfg
Cloning into 'tfg'...
remote: Counting objects: 51, done.
remote: Total 51 (delta 0), reused 0 (delta 0), pack-reused 51
Unpacking objects: 100% (51/51), done.
Checking connectivity... done.
jack@jack-ISM:~/dir$ cd tfg/WhatAClass
jack@jack-ISM:~/dir/tfg/WhatAClass$ git clone https://github.com/Jazriel/WhatAClass
Cloning into 'WhatAClass'...
remote: Counting objects: 1158, done.
remote: Compressing objects: 100% (42/42), done.
remote: Total 1158 (delta 2), reused 22 (delta 2), pack-reused 1114
Receiving objects: 100% (1158/1158), 8.41 MiB | 2.81 MiB/s, done.
Resolving deltas: 100% (662/662), done.
Checking connectivity... done.
jack@jack-ISM:~/dir/tfg/WhatAClass$

```

Figura D.1: Clonación de los repositorios

Obtención código fuente (Clonación del repositorio)

Con cualquier herramienta básica de git se puede clonar un repositorio, en algunas la opción se llama importar. En esta sección se expone cómo hacerlo en línea de comandos. Cabe considerar que los comandos son de linux, para hacerlo en otros sistemas operativos podemos usar sus equivalentes.

```

$ mkdir <DIR>
$ cd <DIR>
$ git clone https://github.com/Jazriel/tfg
$ cd tfg
$ git clone https://github.com/Jazriel/WhatAClass

```

Entorno virtual e instalación de las dependencias

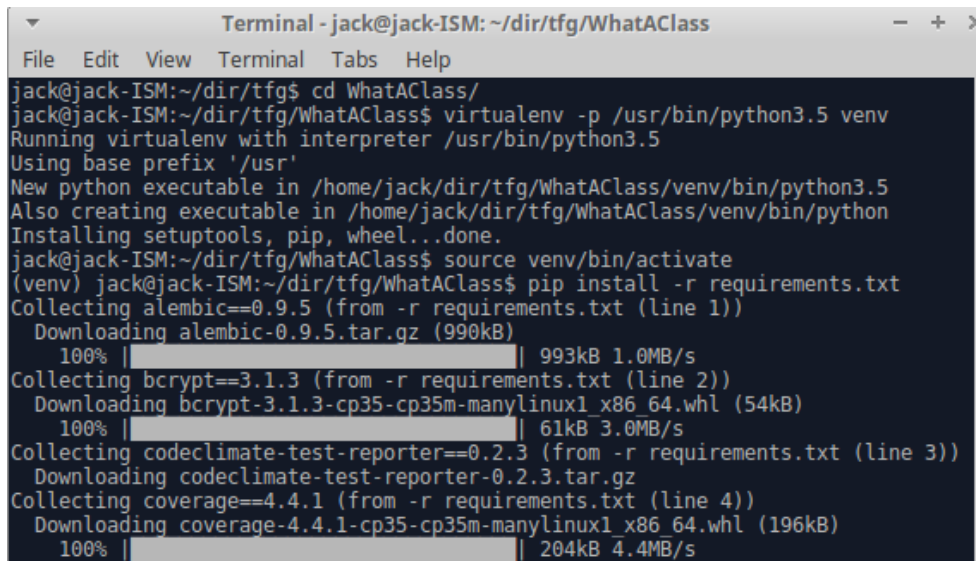
Una vez tenemos el código fuente es muy recomendable instalar las dependencias en un entorno virtual para evitar conflictos con otros proyectos.

```

$ cd WhatAClass
$ virtualenv -p /usr/bin/python<VERSION> <DIR>
$ source <DIR>/bin/activate
$ pip install -r requirements.txt

```

Otra forma de instalar las dependencias es usando el archivo 'setup.py', este archivo está pensado para poder usar pip como sistema de distribución

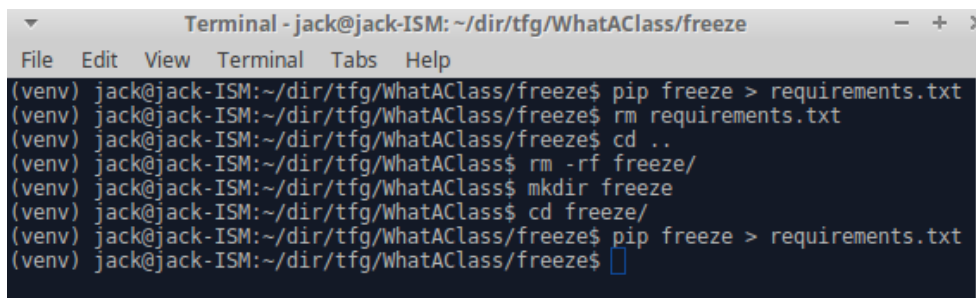


```

Terminal - jack@jack-ISM: ~/dir/tfg/WhatAClass
File Edit View Terminal Tabs Help
jack@jack-ISM:~/dir/tfg$ cd WhatAClass/
jack@jack-ISM:~/dir/tfg/WhatAClass$ virtualenv -p /usr/bin/python3.5 venv
Running virtualenv with interpreter /usr/bin/python3.5
Using base prefix '/usr'
New python executable in /home/jack/dir/tfg/WhatAClass/venv/bin/python3.5
Also creating executable in /home/jack/dir/tfg/WhatAClass/venv/bin/python
Installing setuptools, pip, wheel...done.
jack@jack-ISM:~/dir/tfg/WhatAClass$ source venv/bin/activate
(venv) jack@jack-ISM:~/dir/tfg/WhatAClass$ pip install -r requirements.txt
Collecting alembic==0.9.5 (from -r requirements.txt (line 1))
  Downloading alembic-0.9.5.tar.gz (990kB)
    100% |#####| 993kB 1.0MB/s
Collecting bcrypt==3.1.3 (from -r requirements.txt (line 2))
  Downloading bcrypt-3.1.3-cp35-cp35m-manylinux1_x86_64.whl (54kB)
    100% |#####| 61kB 3.0MB/s
Collecting codeclimate-test-reporter==0.2.3 (from -r requirements.txt (line 3))
  Downloading codeclimate-test-reporter-0.2.3.tar.gz
Collecting coverage==4.4.1 (from -r requirements.txt (line 4))
  Downloading coverage-4.4.1-cp35-cp35m-manylinux1_x86_64.whl (196kB)
    100% |#####| 204kB 4.4MB/s

```

Figura D.2: Creación de entorno virtual e instalación de dependencias



```

Terminal - jack@jack-ISM: ~/dir/tfg/WhatAClass/freeze
File Edit View Terminal Tabs Help
(venv) jack@jack-ISM:~/dir/tfg/WhatAClass/freeze$ pip freeze > requirements.txt
(venv) jack@jack-ISM:~/dir/tfg/WhatAClass/freeze$ rm requirements.txt
(venv) jack@jack-ISM:~/dir/tfg/WhatAClass/freeze$ cd ..
(venv) jack@jack-ISM:~/dir/tfg/WhatAClass$ rm -rf freeze/
(venv) jack@jack-ISM:~/dir/tfg/WhatAClass$ mkdir freeze
(venv) jack@jack-ISM:~/dir/tfg/WhatAClass$ cd freeze/
(venv) jack@jack-ISM:~/dir/tfg/WhatAClass/freeze$ pip freeze > requirements.txt
(venv) jack@jack-ISM:~/dir/tfg/WhatAClass/freeze$

```

Figura D.3: Prueba de uso de pip freeze

del proyecto. No se ha visto necesario el subir el archivo a pip ya que no proporciona un servicio directo, sino que ayuda a proporcionar servicios.

Para comprobar como funciona pip freeze para la actualización de dependencias, se expondrá a continuación un ejemplo ejecutado como continuación del anterior.

```

$ mkdir freeze
$ cd freeze
$ pip freeze > requirements.txt

```

Para actualizar las dependencias, como ya se ha comentado, solo debemos ejecutar este último comando y luego transcribirlo a 'requirements.txt', normalmente con la *pipe* igual que en el ejemplo anterior.

Docker

```
$ docker build .
```

Para lanzar un contenedor y ponerlo en ejecución se puede hacer con el siguiente comando:

```
$ docker run -i -t <image_id>
```

O el siguiente, si queremos usar el nombre del contenedor:

```
$ docker run -i -t <image_name>:<version>
```

Si no nos acordásemos de que imágenes o que nombres tienen, ya que es difícil de acordarnos tanto de las ids como de los nombres auto-generados, podemos usar:

```
$ docker images
```

Para todos estos comandos existen muchas variaciones, pero para su uso básico esta información es suficiente.

Docker compose

Para coordinar varios contenedores en el momento que queramos ejecutar la aplicación debemos usar un servicio de orquestación, como ya estamos usando docker, la opción más fácil y, de momento, profesional es docker compose.

La forma de usar docker compose es con un archivo ‘docker-compose.yml’. En el se especifican todas las configuraciones adicionales que se necesiten.

Una vez tenemos nuestro archivo ‘docker-compose.yml’ podemos usar los siguientes comandos para hacer la construcción de los contenedores y para desplegarlos:

```
$ docker-compose build
$ docker-compose up
```

D.3. Instalación para uso local

Esta instalación está pensada para usuarios avanzados que quieran usar el servicio en local.

1. Se debe comprobar que el ordenador permita la virtualización.
2. Instalar un navegador.
3. Instalar **Docker CE**¹
4. Descargar este archivo **Docker compose**².

¹Docker CE: <https://docs.docker.com/engine/installation/>.

²Docker compose: <https://raw.githubusercontent.com/Jazriel/TFG/master/docker-compose-hub/docker-compose.yml>.

5. Ejecutar el siguiente comando en un lugar con acceso a internet.

```
$ docker-compose pull
```

6. Levantar el servicio con el comando posterior cuando queramos ejecutarlo.

```
$ docker-compose up
```

Si ejecutamos el 6 con internet se ejecutan tanto el 5 como el 6 ahorrando un mínimo de trabajo.

Cabe destacar que esta instalación es igual para windows y para linux.

Tras este proceso basta con acceder a 127.0.0.1 en el navegador.

Apéndice *E*

Documentación técnica de programación

E.1. Introducción

El proyecto, al estar descompuesto en microservicios, tiene varias partes bien diferenciadas.

E.2. Estructura de directorios

Aplicación web

La aplicación web tiene una estructura de directorios algo compleja, por lo que algunos elementos se han omitido del esquema.

WhatAClass/	<i>Principal localización para nuestro código fuente</i>
blueprints/	<i>Localización de los controladores</i>
oauth/	<i>Controladores de OAuth</i>
static/	<i>Archivos estáticos (bootstrap y favicon)</i>
templates/	<i>Plantillas de Jinja2 para generar html o javascript</i>
tensorflow_mng/	<i>Plantillas para tensorflow</i>
user_mng/	<i>Plantillas para el control de usuarios</i>
email/	<i>Plantillas para los email</i>
translations/	<i>Mensajes traducidos (compilación)</i>
utils/	<i>Utilidades</i>
email.py	<i>Utilidad de email</i>
app.py	<i>Builder</i>

APÉNDICE E. DOCUMENTACIÓN TÉCNICA DE PROGRAMACIÓN 29

extensions.py	<i>Extensiones de flask</i>
forms.py	<i>Formularios</i>
models.py	<i>Modelos</i>
Alembic/	<i>Versionado de la DB</i>
versions/	<i>Scripts de migración entre versiones de la DB</i>
babel/	<i>Mensajes con sus traducciones</i>
config/	<i>Configuración de la aplicación</i>
default.py	<i>Configuración por defecto</i>
docs/	<i>Documentación autogenerada</i>
report/	<i>Documentación manual</i>
resources/	<i>Recursos como claves rsa</i>
tests/	<i>Tests de la aplicación</i>
.coveragerc	<i>Que código se ignora por el cubrimiento</i>
.dockerignore	<i>Que ignora docker</i>
.gitignore	<i>Que ignora git</i>
.travis.yml	<i>Configuración de travis</i>
babel.cfg	<i>Configuración de babel (internacionalización)</i>
create_db.py	<i>Script de creación de la DB a partir del modelo</i>
docker-compose.yml	<i>Configuración de docker compose (solo webapp)</i>
Dockerfile	<i>Descripción de construcción del contenedor</i>
LICENSE	<i>Licencia</i>
messages.pot	<i>Mensajes traducidos</i>
Procfile	<i>Comandos para heroku</i>
README.md	<i>Readme para Github</i>
requirements.txt	<i>Requisitos para desarrollador y para VersionEye</i>
requirements-prod.txt	<i>Requisitos para despliegue</i>
run.py	<i>Punto de entrada para debug</i>
runtime.txt	<i>Establece la versión de python para heroku</i>
setup.cfg	<i>Alias para test</i>
setup.py	<i>Instalación de las dependencias</i>
start.py	<i>Punto de entrada sin configuración adicional</i>
test.py	<i>Punto de entrada para los test</i>
uwsgi.ini	<i>Configuración para que uwsgi encuentra la aplicación</i>
wsgi.py	<i>Punto de entrada para uwsgi</i>

Microservicio de tensorflow

dataset/	<i>Estructura de archivos en producción para facilitar los test</i>
graphs/	<i>Modelo y clases.</i>
images/	<i>Estructura de archivos de producción para los test</i>
resources/id_rsa.pub	<i>Clave rsa para comunicación con la aplicación web</i>

scripts/	<i>Localización de los scripts</i>
fit_dataset.py	<i>Script para ajustar el modelo a un dataset</i>
maybe_start_retrain.py	<i>Script que comprueba si la petición ya se está ejecutando o no</i>
run_inference.py	<i>Script para inferir la clase de una imagen</i>
test/	<i>Carpeta donde se guardan los test</i>
Dockerfile	<i>Descripción de construcción del contenedor</i>

Docker compose

Hay dos archivos que se han hecho fuera de las dos anteriores y estos son los docker-compose.yml que coordinan ambos servicios.

Uno está en la raíz de ambas estructuras, este hace la construcción del servicio contando con los fuentes.

El otro que está en docker-compose-hub/docker-compose.yml, y este usa los contenedores disponibles en dockerhub en vez de construirlos desde los fuentes.

E.3. Manual del programador

En la siguiente sección se expondrá como se puede continuar desarrollando la aplicación.

Entorno de desarrollo

Los siguientes elementos son necesarios para trabajar en el proyecto de forma efectiva:

- Python 3.5 o 3.6
- Git
- Gestor de paquetes
- Entorno virtual
- IDE o editor de texto
- Docker

Python

El lenguaje de programación más popular en este momento, tanto para web, como para aplicaciones de empresa es Python [4].

El proyecto se inició con Python 3.5 aunque se ha mantenido la compatibilidad con Python 3.6. Debido a algunos cambios que se han hecho en Python 3.6 con los diccionarios, se recomienda esta versión, ya que proporciona un aumento de rendimiento.

Se puede obtener esta versión (3.6) en [2]. Hay que elegir correctamente la versión y sistema operativo. Después de seguir el manual de instalación tendremos Python instalado.

Git

La forma más simple y útil de descargar y trabajar con el código fuente es a través de un sistema de control de versiones como es Git [13].

Este programa permite movernos a través de la evolución que ha tenido el código a lo largo del tiempo.

Una vez lo tengamos instalado, tenemos dos posibilidades: usar un soporte con GUI o usar un soporte en línea de comandos, por supuesto hay algunos programas que lo instalan automáticamente.

De la primera opción están el proporcionado por Git en algunas instalaciones, los plugin o utilidades para IDEs o un programa especializado como GitKraken.

La segunda opción es la que se recomienda porque ayuda a comprender el funcionamiento de Git de una manera más clara.

Gestor de paquetes

El gestor de paquetes es una utilidad considerada necesaria para facilitar la instalación de paquetes (en este contexto significa un conjunto de software), ya que no solo instalamos el código fuente que pedimos sino que también instalamos las dependencias de dicho código.

Las principales herramientas son **Pip**¹, **Setuptools**² y **Wheel**³.

Si hemos instalado Python 2 con la versión 2.7.9+ o Python 3 con la versión 3.4+ ya tienen instalado pip y setuptools.

En linux, ya que el sistema operativo tiene muchas veces tanto la versión 2.x y 3.x de Python para usar pip tendremos que usar pip2 o pip3 para saber que versión específica se está usando.

Uso básico de pip

Para instalar un proyecto específico con pip [7] se hace con:

```
$ pip install <PROJECT>
```

Algunas opciones adicionales son las versiones específicas o compatibles:

```
$ pip install <PROJECT>==<SPECIFIC>
$ pip install <PROJECT>~=<COMPATIBLE>
```

Para actualizar un proyecto:

¹Pip: <https://pypi.python.org/pypi/pip>.

²Setuptools: <https://pypi.python.org/pypi/setuptools>.

³Wheel: <https://pypi.python.org/pypi/wheel>.

```
$ pip install --upgrade <PROJECT>
```

Por último, cabe destacar que para un conjunto de dependencias la manera más simple de instalar todas es tenerlas en un archivo `requirements.txt`, para instalar a partir de ese archivo se usa el siguiente comando:

```
$ pip install -r requirements.txt
```

Para obtener este archivo el desarrollador anterior debe haber generado ese archivo usando `pip`, esto descargará las dependencias que tiene el entorno desde el que se ha generado.

```
$ pip freeze
```

Entorno virtual

Usar un entorno virtual no es una necesidad a la hora de desarrollar programas en Python, pero se recomienda encarecidamente debido a que eso nos permite encapsular nuestras dependencias. La excepción a la regla anterior es el caso de desarrollar dos programas con dependencias a las mismas bibliotecas en diferentes versiones.

Algunos de los gestores de entornos virtuales más usados son `venv`, `virtualenv` y `conda`. Hay que tener en cuenta que los IDEs necesitan configuración adicional para trabajar con entornos virtuales.

Las características principales de las opciones anteriores son:

- **Venv**: viene incluido en la biblioteca standard de Python, no soporta Python 2.x.
- **Virtualenv**: es la biblioteca más recomendada por los principales tutoriales de Python y se encuentran disponibles envoltorios que facilitan o implementan funcionalidades adicionales, soporta Python 2.6+ y 3.3+.
- **Conda**: no solo es un gestor de entornos virtuales, sino que también gestiona paquetes, soporta también 2.6+ y 3.3+.

Creación de un entorno virtual

Para este ejemplo pondremos como ejemplos a `venv` y `virtualenv`. Las palabras que están entre ‘<’ y ‘>’ son variables.

Venv

Para crear el entorno:

```
$ python3 -m venv <DIR>
```

Para activarlo:

```
$ source <DIR>/bin/activate
```


Para desactivarlo:

```
$ deactivate
```

Virtualenv

Virtualenv al no estar incluido en Python deberemos instalarlo, para ello se recomienda tener un gestor de paquetes como pip instalado antes.

Para instalar virtualenv valdrá con:

```
$ pip install virtualenv
```

Para crear el entorno:

```
$ virtualenv -p /usr/bin/python<VERSION> <DIR>
```

Para activarlo: `sudo pip3 install -r requirements.txt`

```
$ source <DIR>/bin/activate
```

Para desactivarlo:

```
$ deactivate
```

IDE o editor de texto

Se recomienda usar un IDE aunque un editor de texto y una línea de comandos pueden hacer la misma función. En caso de usar un IDE, hay que buscar la configuración para que pueda trabajar con el entorno virtual que hayamos creado o vayamos a crear.

Si no estamos usando un IDE, simplemente debemos entrar en el entorno virtual antes de hacer nada relacionado con la ejecución del proyecto.

Importar el proyecto desde Pycharm

Por la separación del proyecto en microservicios se recomienda el no trabajar con un solo proyecto, se recomienda importar cada servicio en un proyecto y así mantenerlos separados, esto facilita el uso de entornos virtuales.

1. Abrir PyCharm
2. Desde la ventana de bienvenida, hacer click en abrir y abrir la localización `tfg/WhatAClass`
3. Esperar a que PyCharm lea todos los archivos
4. Ir abriendo los cuadros de dialogo y comprobando su importancia

PyCharm ya está preparado para trabajar con git, pip, entornos virtuales y docker, pero muchas veces hace falta configurar estas herramientas. Para configurarlas los cuadros de dialogo son muy utiles ya que proporcionan una

buena cantidad de ayuda, facilitando la instalación de dependencias no instaladas, permitiendo tener docker como opción de ejecución predeterminada, llevando el control de versiones por ti y avisando si algo no sale como se espera.

Por último, tenemos que añadir como *script* de depuración ‘run.py’ y como *script* de prueba ‘test.py’ o la carpeta de ‘test/’ como objetivo.

Añadir características

Para añadir características al programa debemos, en primer momento plantearnos que funcionalidad proveen, si cuadra dentro de una de las categorías que tenemos definidas (home, control de usuarios y tensorflow) o no.

Si cuadra con alguna de las categorías definidas, nos será suficiente con añadir más rutas disponibles dentro de la blueprint de esa sección. Si no cuadra, lo que tendremos que hacer será crear esa blueprint que vayamos a implementar.

Sea cual sea la opción que elijamos acabaremos creando funciones decoradas con ‘@route’ o ‘@blueprint.route’. Estas funciones serán nuestros controladores, lo que devuelvan será lo que vea el cliente, principalmente deberíamos devolver la función ‘render_template’ invocada con la plantilla correspondiente. Para la internacionalización debemos importar de ‘flask_babel’ la función ‘gettext’, normalmente se importa como ‘_’ de manera que el texto internacionalizable pasa antes por esa función.

Como ejemplo de la mayoría de conocimientos requeridos para implementar una funcionalidad nueva se expone la función de login, excesivamente comentada, a continuación.

```

1  from flask import (Blueprint, flash, render_template,
2                        url_for, redirect)
3  from flask_login import current_user
4
5  from flask_babel import gettext as _
6
7  from WhatAClass.forms import LoginForm
8  from WhatAClass.models import User
9
10 user_mng = Blueprint('user_mng', __name__)
11
12 # El decorador que da una ruta de acceso
13 @user_mng.route('/login', methods=['GET', 'POST'])
14 def login():
15
16     # Ruta desde la carpeta templates hasta el archivo
17     # de la vista
18     view = 'user_mng/login.html'
19

```

```

20     # Usuario actual
21     if current_user.is_authenticated:
22         # Función _ para permitir internacionalización
23         flash(_('There is a logged in user already.'))
24         return redirect(url_for('index.base'))
25
26     form = LoginForm()
27
28     # Si el formulario esta siendo subido (POST)
29     if form.validate_on_submit():
30
31         # Hacemos una petición a la base de datos
32         user = User.query.filter_by(
33             email=form.email.data).first()
34
35         # Intentamos hacer login al usuario
36         if not check_and_login(
37             form.password.data, user):
38
39             return render_template(view, form=form)
40
41         flash(_('Logged in successfully.'))
42         return redirect(url_for('index.base'))
43
44     # Salida de una petición GET
45     return render_template(view, form=form)

```

Como se puede ver, lo que en otras tecnologías es una clase, lo que aquí es la *blueprint*, esto hace que no sean tan útiles los diagramas uml.

Si se desea usar git para integrar el cambio, hacer un fork del proyecto y luego una pull request sería lo más adecuado.

Compilación

Python es un lenguaje interpretado y no necesita compilación previa, pero algunas partes de nuestro proyecto sí que pueden necesitar ser compiladas. De momento solo las traducciones necesitan ser compiladas.

Babel

Para traducir con Babel, como ya se ha introducido anteriormente, usaremos gettext y `_`. Una vez todo nuestro texto este en llamadas a esas funciones solo nos queda extraerlo para traducirlo, generar el catálogo del idioma y compilar ese catálogo para que sea de uso más eficiente.

A continuación se expone como se lleva a cabo este proceso:

```
$ pybabel extract -F babel.cfg -o messages.pot WhatAClass/
```

Este primer comando lo que lleva a cabo es la extracción de todos los mensajes a ser traducidos al archivo ‘messages.pot’. Debemos abrir ese archivo con un editor de texto o nuestra IDE para añadir las traducciones correspondientes. Una vez estén añadidas, es hora de generar el catálogo del idioma correspondiente.

```
$ pybabel init -i messages.pot
               -d WhatAClass/translations
               -l es
```

Con esto estamos especificando que la entrada es messages.pot, la salida estará en WhatAClass/translations y el lenguaje es el español, para otros idiomas habría que buscar sus extensiones. Por último, se compilan las traducciones para que babel pueda usarlas de manera eficiente

```
$ pybabel compile -d WhatAClass/translations
```

Docker

Como ya se ha hablado Docker [12, 9, 5] se usa para facilitar el despliegue en cualquier servidor que lo tenga instalado. Para poder disfrutar las ventajas que nos da el uso de este servicio de virtualización debemos configurar el contenedor cómo si fuese un sistema operativo normal y corriente. Esto se lleva a cabo en un archivo llamado ‘Dockerfile’.

Cómo no sería rentable invertir el tiempo de instalar un sistema operativo en un contenedor, la opción que nos da Docker es el equivalente en git a hacer un fork, replica todo el sistema que pidamos y a partir de él podemos usar el sistema como queramos. Cómo se hace esto se verá a continuación explicado en las siguientes secciones.

Antes de poder ejecutar un contenedor debemos construirlo, para ello debemos situarnos en la carpeta que contenga nuestro proyecto, tras eso ejecutaremos el comando siguiente:

```
$ docker build . [-t <name>[:<version>]]
```

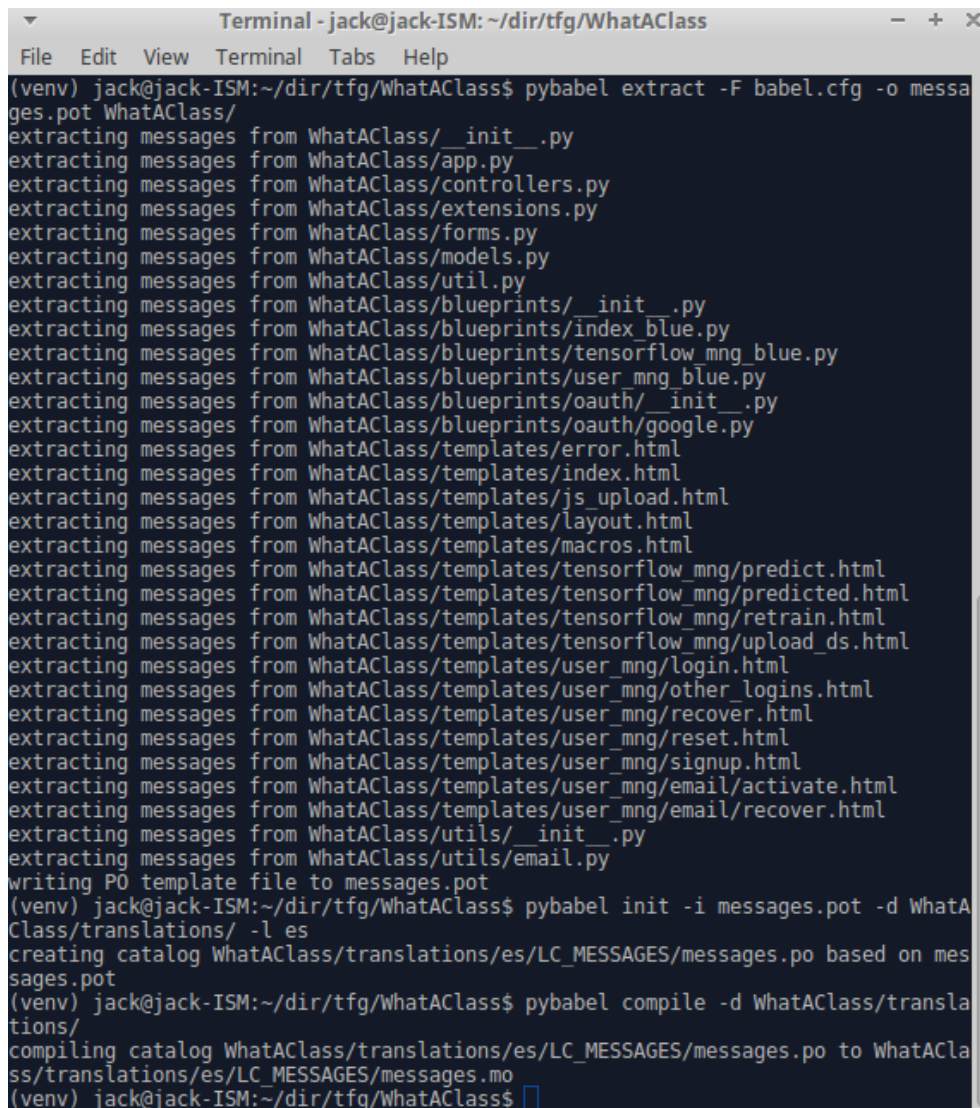
Si por cualquier razón no estuviese el ‘Dockerfile’ en la carpeta raíz del proyecto, deberíamos cambiar el ‘.’ por la ruta hasta el directorio.

Para lanzar un contenedor y ponerlo en ejecución se puede hacer con el siguiente comando:

```
$ docker run -i -t <image_id>
```

O el siguiente, si queremos usar el nombre del contenedor:

```
$ docker run -i -t <image_name>:<version>
```



```

Terminal - jack@jack-ISM: ~/dir/tfg/WhatAClass
File Edit View Terminal Tabs Help
(venv) jack@jack-ISM:~/dir/tfg/WhatAClass$ pybabel extract -F babel.cfg -o messages.pot WhatAClass/
extracting messages from WhatAClass/__init__.py
extracting messages from WhatAClass/app.py
extracting messages from WhatAClass/controllers.py
extracting messages from WhatAClass/extensions.py
extracting messages from WhatAClass/forms.py
extracting messages from WhatAClass/models.py
extracting messages from WhatAClass/util.py
extracting messages from WhatAClass/blueprints/__init__.py
extracting messages from WhatAClass/blueprints/index_blue.py
extracting messages from WhatAClass/blueprints/tensorflow_mng_blue.py
extracting messages from WhatAClass/blueprints/user_mng_blue.py
extracting messages from WhatAClass/blueprints/oauth/__init__.py
extracting messages from WhatAClass/blueprints/oauth/google.py
extracting messages from WhatAClass/templates/error.html
extracting messages from WhatAClass/templates/index.html
extracting messages from WhatAClass/templates/js_upload.html
extracting messages from WhatAClass/templates/layout.html
extracting messages from WhatAClass/templates/macros.html
extracting messages from WhatAClass/templates/tensorflow_mng/predict.html
extracting messages from WhatAClass/templates/tensorflow_mng/predicted.html
extracting messages from WhatAClass/templates/tensorflow_mng/retrain.html
extracting messages from WhatAClass/templates/tensorflow_mng/upload_ds.html
extracting messages from WhatAClass/templates/user_mng/login.html
extracting messages from WhatAClass/templates/user_mng/other_logins.html
extracting messages from WhatAClass/templates/user_mng/recover.html
extracting messages from WhatAClass/templates/user_mng/reset.html
extracting messages from WhatAClass/templates/user_mng/signup.html
extracting messages from WhatAClass/templates/user_mng/email/activate.html
extracting messages from WhatAClass/templates/user_mng/email/recover.html
extracting messages from WhatAClass/utis/__init__.py
extracting messages from WhatAClass/utis/email.py
writing PO template file to messages.pot
(venv) jack@jack-ISM:~/dir/tfg/WhatAClass$ pybabel init -i messages.pot -d WhatAClass/translations/ -l es
creating catalog WhatAClass/translations/es/LC_MESSAGES/messages.po based on messages.pot
(venv) jack@jack-ISM:~/dir/tfg/WhatAClass$ pybabel compile -d WhatAClass/translations/
compiling catalog WhatAClass/translations/es/LC_MESSAGES/messages.po to WhatAClass/translations/es/LC_MESSAGES/messages.mo
(venv) jack@jack-ISM:~/dir/tfg/WhatAClass$

```

Figura E.1: Proceso de traducción

Si no nos acordásemos de que imágenes o que nombres tienen, ya que es difícil de acordarnos tanto de las ids como de los nombres auto-generados, podemos usar:

```
$ docker images
```

Para todos estos comandos existen muchas variaciones, pero para su uso básico esta información es suficiente.

Aplicación web

En el siguiente ejemplo se muestra un archivo Dockerfile, específicamente el de la aplicación web, lo que hace cada línea se verá comentado en el propio código.

Dado que necesita usar una base de datos que no se lanza en el mismo contenedor no podemos usar este archivo de forma individual. Si quisiésemos poder usarlo de forma individual, podríamos configurar un volumen, que es la unidad de almacenamiento persistente de docker, para almacenar una base de datos SQLite, esto se podría añadir en los archivos de configuración para que funcionase.

```
1 FROM tiangolo/uwsgi-nginx-flask:flask-python3.5
2 # De que <usuario>/<imagen>:<version> estamos 'heredando'
3 # Es un link a dockerhub de manera implícita.
4
5 MAINTAINER Javier Martinez "javyermartinez@gmail.com"
6 # Quién mantiene la imagen.
7
8 WORKDIR /app
9 # Directorio de trabajo,
10 # el equivalente a cd /app de linux
11
12 # Copiamos el código fuente
13 COPY ./WhatAClass ./WhatAClass
14
15 # Copiamos la parte estática a una carpeta
16 # especialmente configurada para que nginx
17 # pueda servirlo más rápidamente
18 COPY ./WhatAClass/static ./static
19 # Copy es un comando que copia desde nuestra maquina y
20 # directorio hasta el contenedor y directorio de trabajo.
21
22 RUN mkdir -p ./static/images
23 # Podemos ejecutar comandos del sistema operativo
24 # con run, en este caso creamos un directorio.
25
```

```

26 # {...
27 # ...}
28
29 # Añadimos una variable de estado con env
30 ENV PYTHONPATH '$PYTHONPATH:
31                 /WhatAClass:
32                 /WhatAClass/WhatAClass'
33
34 # Instalamos las dependencias
35 RUN pip3 install --editable .

```

Tensorflow

El contenedor de tensorflow es heredado de una instalación de tensorflow, a partir de ese punto añadimos un daemon de ssh.

En este archivo tenemos un par de comandos distintos a los del otro que explicaremos a continuación.

```

1
2 FROM gcr.io/tensorflow/tensorflow:latest-devel-py3
3 # Imagen de tensorflow y con un link explícito a gcr.io
4
5 RUN apt-get update && apt-get install -y openssh-server
6 # Instalar el daemon de ssh
7
8 # {...
9 # ...}
10
11 EXPOSE 22
12 # Exponemos un puerto para que se pueda
13 # acceder a el más fácilmente
14
15 CMD ["/usr/sbin/sshd", "-D"]
16 # CMD es parecido a run, pero en algunos casos
17 # simplifica la operación.

```

Docker compose

Para coordinar ambos contenedores en el momento que queramos ejecutar la aplicación debemos usar un servicio de orquestación, como ya estamos usando docker, la opción más fácil y, de momento, profesional es docker compose.

La forma de usar docker compose es con un archivo ‘docker-compose.yml’. En el se especifican todas las configuraciones adicionales que se necesiten. A continuación se mostraran un par de ejemplos del mismo.

```

1 # Versión del archivo docker compose
2 # para permitir retrocompatibilidad
3 version: '3.0'
4
5 # Servicios que se desplegarán
6 services:
7   # Nombre arbitrario del servicio
8   web:
9     # Directorio de construcción
10    build: ./
11    # Puertos
12    ports:
13      - "80:80"
14    # Redes internas a las que puede acceder
15    networks:
16      - backend
17    # Servicios de los que depende, no funciona
18    # suficientemente bien.
19    depends_on:
20      - database
21    # Variables de entorno
22    environment:
23      # Como se puede ver tras el @ accedemos a
24      # database como una dirección de red, esto
25      # se debe a que su dirección interna es el
26      # nombre de su servicio.
27      - DATABASE_URL=postgresql://postgres@database #{...}
28   database:
29     # Imagen que usamos ya que no la construimos.
30     image: "postgres"
31     networks:
32       - backend
33 # Redes existentes
34 networks:
35   backend:

```

‘depends_on’ no funciona de forma correcta ya que no espera a ninguna señal del servicio del que depende. Por lo que los servicios no tienen por qué estar preparados para recibir clientes, pudiendo causar errores. Esto llega a ser un problema sobre todo en los sistemas que pueden tardar mucho en encenderse como las bases de datos.

Se ha solucionado añadiendo una espera de hasta 25 segundos en la aplicación web para la base de datos.

Otro ejemplo de docker compose algo más complejo, pero casi idéntico:

```

1 version: '3.0'
2 services:
3   web:
4     # {...}
5     environment:
6       - DATABASE_URL=postgresql:// # {...}
7       # Se puede cambiar el despliegue de forma simple
8       # mediante variables de entorno, si se mantiene la
9       # configuración.
10      - WORKER_HOST_NAME=worker
11      - WORKER_PORT=22
12      - WORKER_USER=root
13      - WORKER_PASSWORD=screencast
14      - TENSORFLOW=True
15     volumes:
16       # Donde se monta el volumen
17       # <nombre_del_volumen>:<directorio donde se monta>
18       - images:/app/static/images
19
20   database:
21     # {...}
22   worker:
23     # {...}
24 networks:
25   backend:
26 # Volúmenes compartidos
27 volumes:
28 # Volumen con nombre
29 images:

```

Una vez tenemos el archivo ‘docker-compose.yml’ podemos usar los siguientes comandos para hacer la construcción de los contenedores y para desplegarlos:

```

$ docker-compose build
$ docker-compose up

```

Este proceso es lento y a no ser que queramos probar como se integran los servicios no conviene usar este método de depuración.

E.4. Despliegue

Para hacer un despliegue en un lugar en el que no este disponible el código fuente la manera más simple es instalar docker y ejecutar un comando de los dos siguientes:

```
$ wget 'URL' -O docker-compose.yml && docker-compose up
```

```
$ curl 'URL' >> docker-compose.yml && docker-compose up
```

Siendo `'https://raw.githubusercontent.com/Jazriel/TFG/master/docker-compose-hub/docker-compose.yml'` la url.

Si ya tuviésemos ese archivo descargado bastaría con ejecutar `'docker-compose up'` en el mismo directorio del archivo.

E.5. Pruebas del sistema

Las pruebas se pueden ejecutar con `test.py` que lanza los tests de la carpeta `tests/`. Si queremos ver el recubrimiento del código, podemos usar una herramienta como `coverage` y ejecutar `test.py` a traves de esa herramienta.

E.6. Configuración

La configuración esta pensada para poder ser diferente en cada despliegue desde el archivo `'docker-compose.yml'`, esto se logra importando variables de entorno con valores por defecto si la variable de entorno no estuviese definida. Los valores por defecto se configuran en el archivo `'config/default.py'`.

Base de datos

La base de datos necesita solo la uri para estar configurada, principalmente la forma de una de este tipo es:

```
postgresql://scott:tiger@localhost:5432/mydatabase
```

Las distintas partes a tener en cuenta son:

- Protocolo: la mayoría de bases de datos cuentan con uno propio con su nombre. En este caso *postgresql* es la base de datos y el protocolo.
- Usuario: esta vez *scott*
- Contraseña: *tiger*
- Host: dirección con la que nos comunicamos. Hay que tener en cuenta que si usamos Docker Compose la red interna cuenta con un DNS de manera que podemos especificar el nombre del otro servicio. En esta ocasión será: *localhost*

- Puerto: puerto a usar para la conexión. *5432* es el puerto por defecto de PostgreSQL.
- Base de datos: por supuesto se pueden tener varias bases de datos en un servidor por lo que tenemos que proporcionar el nombre de la que nos interesa. Aquí es *mydatabase*.

Como alternativa a tener una base de datos alojada en un servidor tenemos SQLite, que es una base de datos que se puede incrustar, al contrario que muchas otras alternativas. En este caso tendremos que tener en cuenta que SQLite necesita permisos de lectura y de escritura del archivo de la base de datos y del directorio en el que se sitúa dicho archivo.

Las direcciones de SQLite son algo diferentes al ser una base de datos incrustada.

```
sqlite://///path/to/the/database.db
```

Simplemente son la dirección de la base de datos en el sistema de archivos. Una consideración a tener en cuenta es que usan cuatro '/' una de ellas escapa la raíz del sistema de archivos.

La variable usada es *DATABASE_URL*.

OAuth

Para configurar *OAuth* para una aplicación debemos pedir un *token* y un secreto, esto en Google se hace a través de la sección de desarrolladores o **Google APIs**⁴. Estos valores se especifican en 'config/default.py' tienen los nombres *GOOGLE_CLIENT_ID* y *GOOGLE_SECRET*.

Servidor de correo electrónico

Para configurar el correo electrónico debemos comprobar que el servidor permite el uso de SMTP para la comunicación, una vez está configurado de esta manera debemos proporcionar:

- Dirección propia (usuario@provedor.exten)
- Contraseña
- Dirección del servidor
- Puerto

Estos se proporcionan en las siguientes variables: *EMAIL_FROM*, *EMAIL_PASS*, *EMAIL_HOST* y *EMAIL_PORT*.

⁴Google APIs: <https://console.developers.google.com/>.

E.7. Subproducto: Seshat

Como subproducto del proyecto se ha integrado la autenticación de usuarios a el proyecto Seshat en el repositorio [SeshatAuth](#)⁵. Este repositorio es privado y se necesita permiso de acceso, aun así se incluye un clon del mismo en las copias digitales por facilidad de acceso.

Este subproducto también se ha subido de forma minimizada al repositorio [auth-that](#)⁶.

Estructura de directorios

La estructura de directorios se ha mantenido, simplemente se han añadido los archivos necesarios. En la carpeta Servidor están todos los archivos añadidos, se comentan solo aquellos archivos que han sido modificados o añadidos.

config/default.py	<i>Archivo donde se almacena la configuración</i>
src/	
server/	
templates/	<i>Se ha modificado el formulario maestro para poder devolver mensajes</i>
user_control/	<i>Localización del código</i>
user_server.py	<i>Principal archivo donde se integra la autenticación.</i>
test/	
.gitignore	<i>Se ignoran los archivos .idea</i>
debug.py	<i>Ejecución en depuración</i>
requirements.txt	<i>Cambio de las dependencias</i>
run.py	<i>Ejecución en producción</i>

Manual del programador

La serie de pasos para pasar de tener el subproducto auth-that y Seshat se detallan a continuación, como consideración los dos proyectos en un principio deberían estar independientes:

1. En auth-that: refactorizar `project_name/` a `src/`, en otros proyectos serían otros nombres.
2. Eliminar los archivos ‘placeholder’ como ‘`src/__init__.py`’ y ‘`src/project.py`’
3. Copiar todo el proyecto auth-that a la raíz del código fuente otro proyecto.

⁵SeshatAuth: <https://github.com/cgosorio/SeshatAuth>.

⁶auth-that: <https://github.com/Jazriel/auth-that>.

4. Cambiar la importación en ‘user_server.py’ para que importe la app de flask de la otra aplicación.
5. Configurar la base de datos(uri), servidor de email y *token* de Google. Esto se puede hacer en ‘config/default.py’ o ‘src/user_server.py’.

Generalmente no debería haber problemas, pero debido a que no se pueden asumir casi nada de que tenía la aplicación que hemos importado puede ser que den problemas, principalmente, dos cosas:

- Extensiones de Flask iguales
- Rutas iguales

Aunque normalmente las extensiones de Flask no tendrían porque dar problemas podrían hacerlo. Tristemente no hay una solución fácil y uniforme para esta clase de problema. Hay que arreglarlo a mano.

Las rutas hay que comprobarlas manualmente, o podemos añadir un prefijo al subproducto de autenticación en ‘user_server.py’:

```

1 from .user_control.controllers import user_mng
2
3 app.register_blueprint(user_mng)
4 # Se cambia esta linea por:
5 app.register_blueprint(user_mng, url_prefix='/user ')
6 # O
7 app.register_blueprint(user_mng, subdomain='user ')
```

La primera direccionará las rutas del tipo: ‘*pagina.com/user/login*’

La segunda lo hará de la siguiente manera: ‘*user.pagina.com/login*’

Ademas de estos pequeños inconvenientes la persona responsable de desplegar el proyecto deberá configurar tanto la base de datos, el servicio de correo electrónico y los valores para OAuth. Todo esto se puede hacer de la misma manera que en la otra aplicación pero el archivo esta situado en ‘Servidor/config/default.py’.

Documentación de usuario

F.1. Introducción

En este anexo mostramos en que sistemas se puede usar la aplicación y bajo qué condiciones de manera que la puedan usar los clientes.

F.2. Requisitos de usuarios

El único requisito que necesita un usuario que quiera usar nuestro sistema es un navegador. Necesitamos *javascript* y *cookies* activados, las *cookies* son para mantener la sesión.

Navegadores con los que se ha comprobado:

1. Firefox 53
2. Chrome 57

F.3. Instalación

Al proporcionar nuestro servicio como una página web no necesitamos que el cliente o usuario instale la aplicación.

Si de todas las maneras se hubiese decidido que cada usuario tiene la aplicación en su ordenador, se pueden seguir las instrucciones del manual de instalación para usuarios avanzados [D.3].

F.4. Manual del usuario

El manual de usuario es más simple que otras aplicaciones, esto se debe a que el enfoque del proyecto no ha sido tanto tener una aplicación con muchas funcionalidades, sino seguir un proceso de desarrollo que permitiera utilizar

algunas de las tecnologías de desarrollo más en boga actualmente. He querido aprovechar la realización del proyecto como una oportunidad de formación y aprendizaje adicional al realizado en el grado, y de este modo mejorar mi preparación de cara a la inserción en el mercado laboral.

El usuario generalmente llegará a la aplicación por su índice o portada. A esta página generalmente se accede con la dirección IP 127.0.0.1 si la ejecución es local.

Para usar la aplicación a partir de ese punto necesita registrarse o iniciar sesión con una de las opciones de inicio de sesión alternativas.

Tras registrarse como usuario o tripulante, si esta preparado el correo, se envía un correo electrónico a su dirección personal.

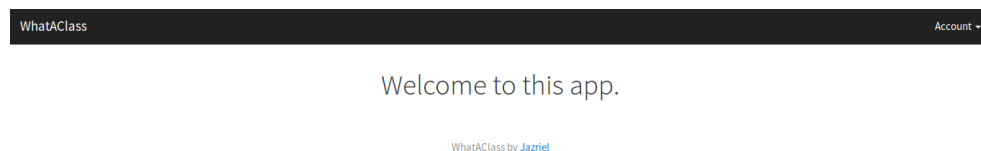
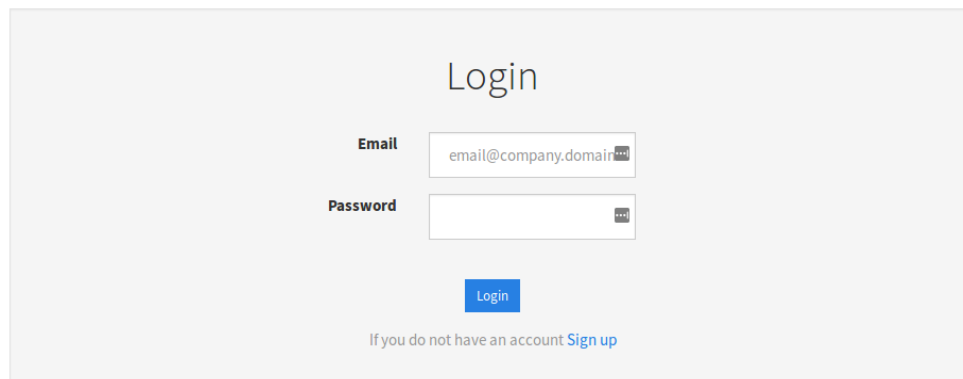


Figura F.1: Punto de entrada a la aplicación

The image shows a registration form titled 'Sign up!'. It has two input fields: 'Email' with the placeholder text 'email@company.domain' and 'Password'. Below the fields is a blue button labeled 'Sign up'.

Figura F.2: Pantalla de registro de usuario



The login screen features a light gray background. At the top center is the title "Login". Below it are two input fields: "Email" with the placeholder text "email@company.domain" and "Password" which is empty. Both fields have a small eye icon on the right. A blue "Login" button is positioned below the password field. At the bottom, a link says "If you do not have an account [Sign up](#)".

Figura F.3: Pantalla de inicio de sesión

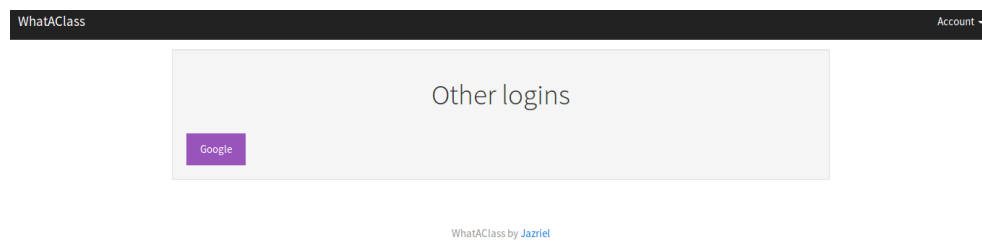


Figura F.4: Inicios de sesión alternativos

Una vez se confirma la recepción del correo, se activa la cuenta, permitiendo así usar servicios que antes no estaban disponibles.

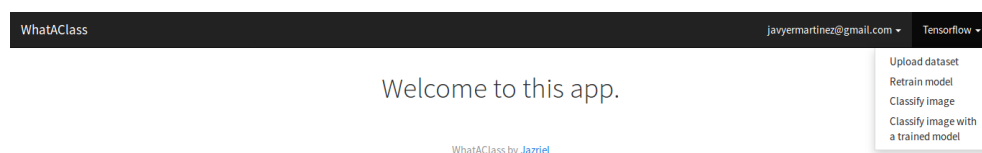


Figura F.5: Cambio de las zonas accesibles al iniciar sesión

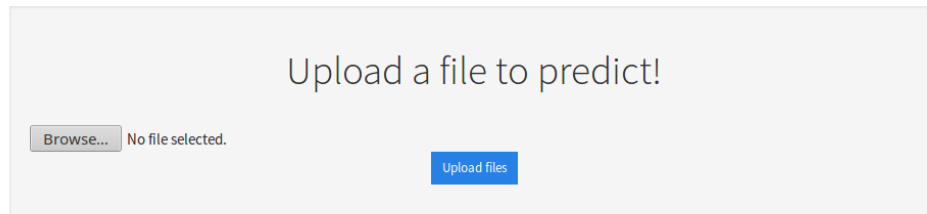


Figura F.6: Pantalla de acceso al servicio de clasificación

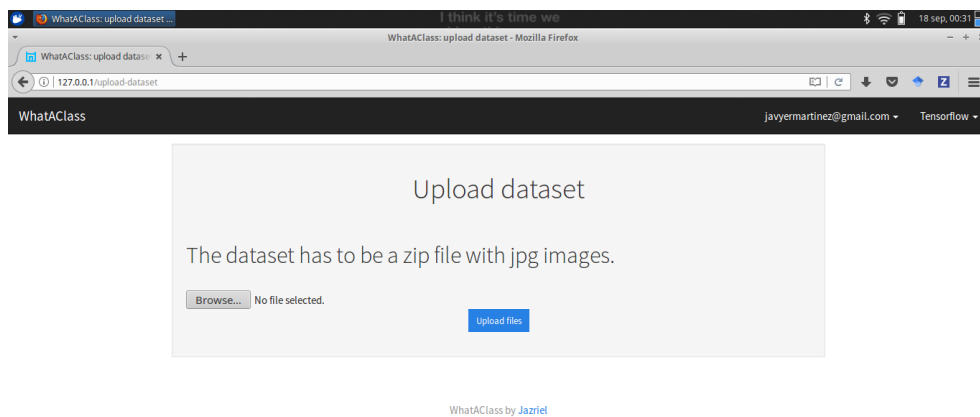


Figura F.7: Pantalla de subida del dataset

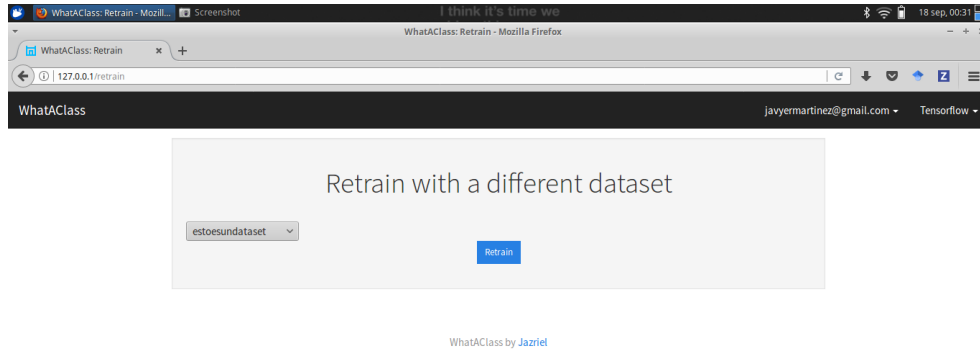


Figura F.8: Pantalla de reentrenamiento del modelo

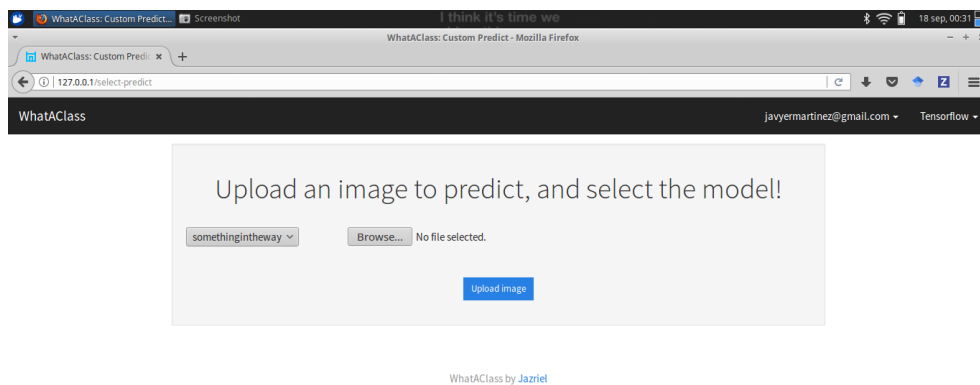


Figura F.9: Pantalla del servicio de clasificación tras el reentrenado

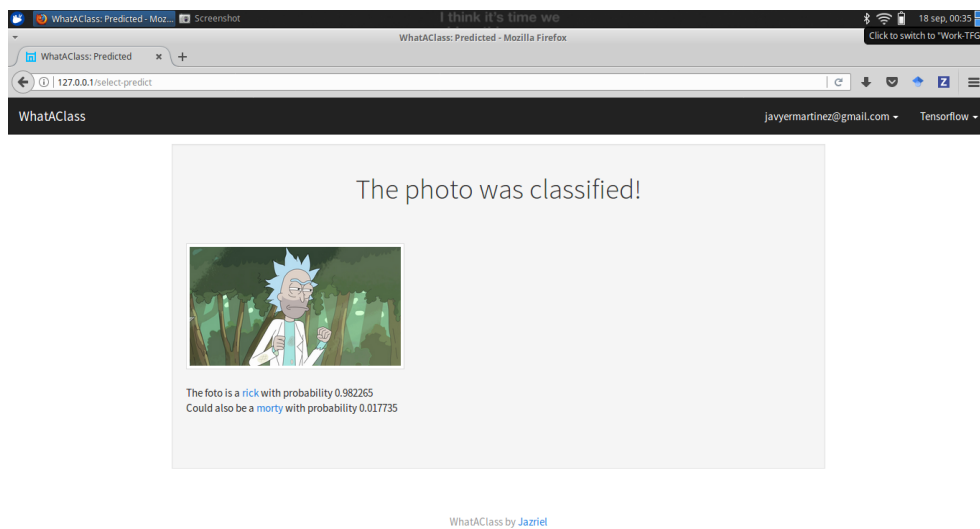


Figura F.10: Pantalla de clasificación tras reentrenado del modelo

Bibliografía

- [1] License recommendations. <https://www.gnu.org/licenses/license-recommendations.html>.
- [2] Python download. <https://www.python.org/downloads/>.
- [3] Gnu lesser general public license, 1999. <https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html>.
- [4] The top programming languages 2017, 2017. <http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2017>.
- [5] Shrikrishna Holla. *Orchestrating docker*. Packt Publishing Ltd, 2015.
- [6] Image-net Team. Image-net. <http://image-net.org/>.
- [7] Equipo de desarrollo PyPA: <https://www.pypa.io/>. pip. <https://pypi.python.org/pypi/pip>.
- [8] Equipo de desarrollo PyPA: <https://www.pypa.io/>. Virtualenv. <https://pypi.python.org/pypi/virtualenv/>.
- [9] Pethuru Raj, Jeeva S Chelladhurai, and Vinod Singh. *Learning Docker*. Packt Publishing Ltd, 2015.
- [10] Jeff Sutherland and JJ Sutherland. *Scrum: the art of doing twice the work in half the time*. Crown Business, 2014.
- [11] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbig-niew Wojna. Rethinking the inception architecture for computer vision. corr abs/1512.00567 (2015), 2015.
- [12] Docker Team. Docker. Docker: <https://www.docker.com/>
Descarga: <https://store.docker.com/search?offering=community&type=edition>.

- [13] Linus Torvalds y la comunidad Open Source. Git. **Git:** <https://git-scm.com/>
Tutorial recomendado (básico y visual): <http://ndpsoftware.com/git-cheatsheet.html>
Descargar git: <https://git-scm.com/downloads>.