UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

título del TFG
Documentación Técnica

Presentado por nombre alumno
en Universidad de Burgos — 3 de julio de 2017
Tutor: nombre tutor

Índice general

Índice de figuras

Índice de tablas

Apéndice A

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 planteo seguir una metodología ágil, esta sería scrum 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. En labores de investigación que no se sabían si iban a ser usables más adelante las cuales no tenían ningún entregable claro se decidió no incluirlas a partir de cierto punto para no acumular demasiada información innecesaria en el repositorio.

En retrospectiva se ha deducido 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 a el 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 las *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 acaban 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 planteo 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 docu-

mentación. 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 planteo la elección del *framework* de *Machine Learning* que usar. La decisión se investigó más entre sklearn, Theano y Tensorflow, aunque de tener la opción ahora se hubiese elegido Keras.

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 cual. 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. Resulto no influir apenas, puede ser que una vez maduren otras alternativas haya que intentar desacerse de *Docker*

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.

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.

Tabla A.1: Costes de personal

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

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 estudiaran 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:

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.

Tabla A.2: Costes de material al mes

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

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 esta 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.

Una vez consigamos calcular tiempos de computación podemos restringir a cada usuario una cantidad de tiempo. 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)[2] , esta licencia esta 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". 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 codigo bajo la licencia que mejor nos parezca o incluso podríamos mantenerlo privado como un Secreto de negocio

(Trade Secret) ya que proporcionamos SaaS (Software as a Service). Basandome en las licencias más comunes open source, 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 otra alternativa más interesante era Creative Commons.

Apéndice B

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 y el conjunto de datos es Imagenet.

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

B.3. Catalogo 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.

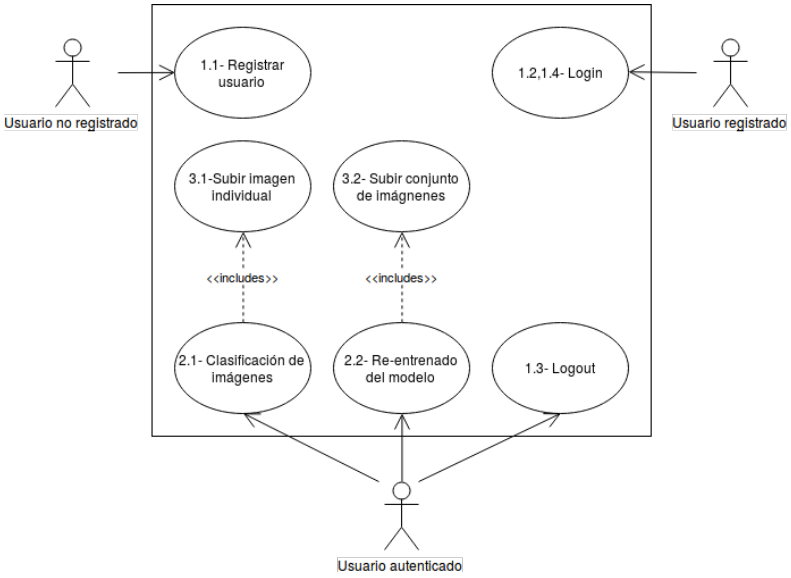

Figura B.1: Diagrama de casos de uso

CU-01	**Registro de usuario**
Autor	Javier Martínez Riberas
Requisitos asociados	RF-1.1
Descripción	Permite al usuario registrarse.
Precondición	Ninguna.
Acciones	1. El usuario rellena el formulario de registro (email y contraseña). 2. Si esta configurada la confirmación por email se necesita confirmar para terminar el registro.
Postcondición	El usuario se registra en la base de datos.
Excepciones	▪ Fallo de la configuración de email. (Si no hay no se da)
Importancia	Alta

CU-01	**Registro de usuario**

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	1. El usuario rellena el formulario de login (email y contraseña). O el equivalente en google.
Postcondición	El usuario consigue una sesión.
Excepciones	▪ 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	1. El usuario hace click el botón de logout.
Postcondición	El usuario cierra una sesión, la cual se elimina.
Excepciones	Ninguna.
Importancia	Alta

CU-03	**Logout**

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	1. El usuario sube una foto. 2. Opcional. El usuario elige un re-entrenamiento específico.
Postcondición	El usuario ve la clase más probable y otras dos probables.
Excepciones	▪ Fallo de formato de imagen.
Importancia	Alta

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

CU-05	**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.

CU-05	**Re-entrenado de modelos**
Acciones	1. El usuario elige un dataset. 2. El usuario debe esperar una cantidad decente de tiempo y tendrá el modelo disponible.
Postcondición	El usuario tras un periodo de tiempo tiene disponible un modelo.
Excepciones	▪ Fallo del formato de las imágenes o de la estructura de archivos.
Importancia	Media-Alta

Tabla B.5: CU-05 Re-entrenado de modelos.

Apéndice C

Especificación de diseño

C.1. Introducción

En este anexo se expone el diseño que se ha usado para llevar a cavo 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:

- **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 *tokenes* 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 de el nombre del usuario y de el nombre del 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 tentemos que tener en cuenta que sólo el modelo del usuario esta dentro de la base de datos de manera que el resto se controla por software y se guarda dentro de los volúmenes de *docker*

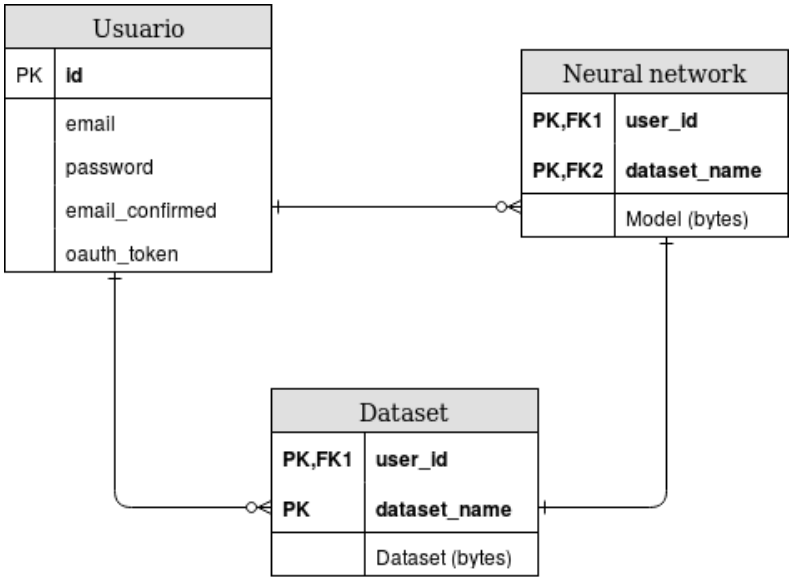

Figura C.1: Diagrama entidad relacción

Paso de datos

El paso de datos entre los servicios ahora mismo se hace a través de SSH, esto se hace ya que nos proporciona seguridad. 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.

C.3. Diseño arquitectónico

El diseño arquitectónico de la aplicación es uno reconocido ya que se usa en muchas páginas web para permitir mayor escalabilidad. 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* antes de las bases de datos para permitir mayor velocidad de acceso. Si quisiésemos replicar micro servicios deberíamos poner un *reverse proxy* o unirlos desde el que ya está.

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.

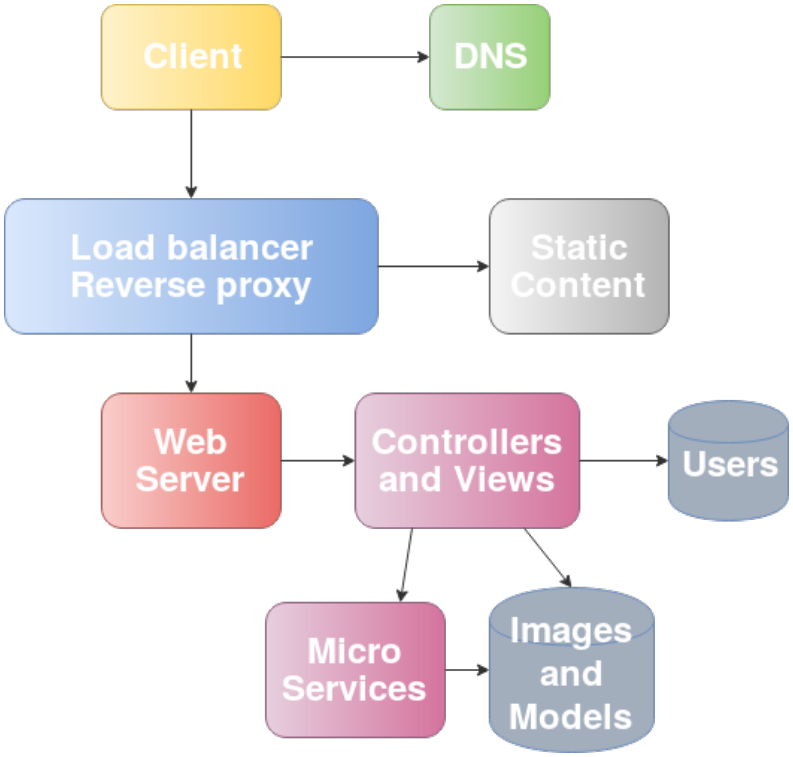

Figura C.2: Arquitectura de la aplicación

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

Factory Method

En la aplicación que hemos creado se ha usado un factory method para permitir la creación de varias instancias de la aplicación de manera 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 las partes del factory method están bien establecidas y un programador que no conozca toda la aplicación puede solo añadir un par de líneas en el método que corresponda.

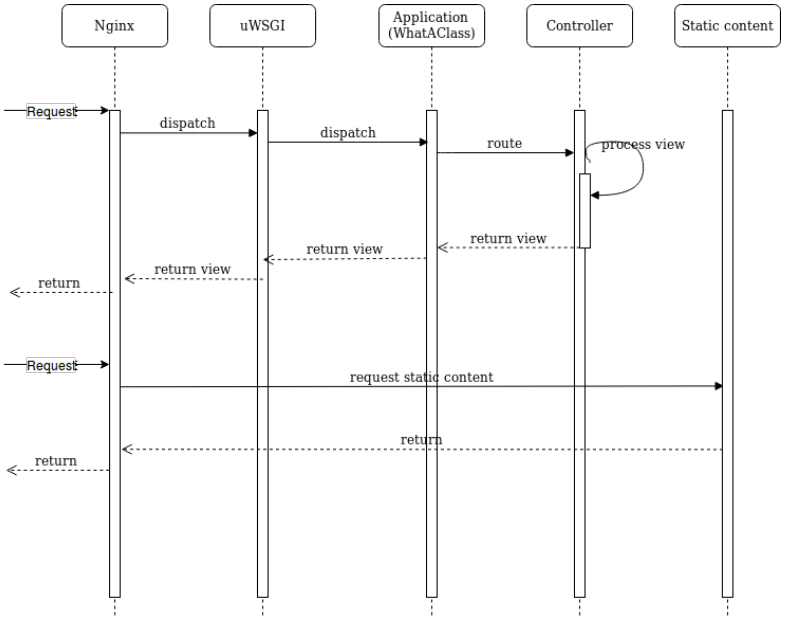

Figura C.3: Diagrama de secuencia

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).

C.5. Diseño de paquetes

Aplicación

La aplicación esta descompuesta en el factory method ("app.py"), extensiones, formularios, modelo, utilidades y controladores.

Los controladores están en la carpeta blueprints pero se incluyen dentro de "controllers.py" para reducir el acoplamiento con el factory method.

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

Configuración

La configuración en general se implementa en la carpeta "config", de 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.

Apéndice D

Documentación técnica de programación

D.1. Introducción

El proyecto al estar descompuesto en microservicios tiene varias partes bien diferenciadas. Esto se refleja en todas las partes del proyecto.

D.2. Estructura de directorios

La estructura de directorios depende del proyecto, en el proyecto web la estructura es:

- alembic: carpeta autogenerada por alembic, control de versiones de la base de datos.
- babel: carpeta que guarda las traducciones generadas por babel.
- config: carpeta con los archivos de configuración, algunos son fuentes en python.
- docs: carpeta donde se guarda lo necesario para ejecutar sphinx (generación de documentacion incode).
- report: carpeta donde reside la documentación out of code.
- tests: tests para la aplicación.
- WhatAClass: carpeta donde se mantienen la mayoría de los fuentes, estos fuentes sirven para contener todas las partes del proyecto, algunos solo direccionan a las carpetas donde están los fuentes .

- WhatAClass/translations: carpeta para las traducciones ya compiladas para no tener que incluirlo en cada ejecución o despliegue.
- WhatAClass/static: para tener archivos que se pueden servir independientemente de manera estática.
- WhatAClass/templates: plantillas a ser interpretadas con jinja2.
- WhatAClass/** : el resto de carpetas se usan para guardar código de una manera más organizada que simplemente no tener carpetas.
- El resto de archivos que se extienden a partir de la raiz del microservicio son para control de versiones, integración continua, despliegue, instalación, tests...

* .coveragerc: recubrimiento de los test.
* .gitignore: control de versiones.
* .travis.yml: integración continua.
* Dockerfile: docker y contenerización.
* Procfile: despliegue en heroku.
* README.md: documentación.
* babel.cfg: configuración de la traducción.
* create_db.py: script para crear la base de datos posiblemente sea eliminado.
* docker-compose.yml: docker compose para despliegue con la base de datos directamente.
* requirements-prod.txt: para instalación.
* requirements.txt: para instalación.
* run.py: ejecución con el servidor que proporciona flask, solo para debug, no usar en producción.
* runtime.txt: heroku, especificación de la versión.
* setup.cfg: instalación con pip, más automática y transparente al usuario.
* setup.py: instalación con pip, más automática y transparente al usuario.
* start.py: script para generar la app, no genera base de datos.

* test.py: script para testear la app.

* uwsgi.ini: configuración para que uwsgi conozca donde esta el script de ejecución en producción.

* wsgi.py: script que genera la aplicación y la base de datos, esta preparado para ser llamado por uwsgi en producción.

D.3. Manual del programador

Se recomienda usar un IDE aunque no es necesario. Con el *script* 'run.py' podemos ejecutar la aplicación para *debug*.

Para añadir cosas a la página web necesitaremos de conocimientos de Flask o de un *framework* web similar, Spring (Java) es similar a como funciona Flask, aunque como cabe esperar tiene diferencias considerables.

Tras conocer Flask debemos conocer sus *blueprints*. Estas son una herramienta que principalmente nos deja descomponer el código en varios apartados permitiendo mantener distintas partes de la aplicación por distintas personas.

Para añadir la *blueprint* a la aplicación nos debemos dirigir a WhatAClass/app.py ya que es el archivo donde esta la factoría de la aplicación. Ya hay ejemplos codificados de esto en 'app.py' y solo tenemos que verlos y los lugares de donde hemos importado esas *blueprints* para saber cómo seguir desarrollando sistemas similares.

D.4. Compilación, instalación y ejecución del proyecto

Se recomienda usar un gestor de entornos virtuales como venv, pyenv, conda...

Para este ejemplo usaremos el recomendado por la documentación oficial de python venv:

Para crearlo: 'python3 -m venv ./venv'

Para activarlo: 'source venv/bin/activate'

Para desactivarlo: 'deactivate'

Los pasos para instalar el servicio web son:

- sudo apt install git python3-pip

- git clone https://github.com/Jazriel/WhatAClass.git

- cd WhatAClass

- sudo pip3 install -r requirements.txt

La instalación se puede hacer desde los fuentes con 'sudo pip3 install -r requirements.txt' o con 'sudo pip3 install –editable .'. Esto usa o requirements.txt o setup.py para instalar las dependencias, ahora mismo se favorece la instalación con el primer comando. Al ser Python un lenguaje interpretado no necesitamos compilarlo manualmente, se compilará JIT (just in time).

Para ejecutar en desarrollo lo mejor sería usar el *script* 'run.py' (*script* específico para desarrollo, no se puede usar en producción), aunque se puede usar un *script* listo para producción no se recomienda ya que los errores son menos descriptivos y mucho más difíciles de solventar.

D.5. Despliegue

Para desplegar esta preparado para ser desplegado con docker cuya instalación se puede encontrar en el siguiente enlace: instalación de docker

Se puede ejecutar con docker pero se recomienda usar docker-compose:

user@machine: /folder$ docker-compose build & & docker-compose up

Esto nos permite saltarnos toda la instalación de programador, ya que todo eso se hace automáticamente.

Hay gente que avoca por tener el entorno de programación dentro de un contenedor. Según las necesidades de la persona y del proyecto en particular esto puede ser una buena idea.

D.6. Pruebas del sistema

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

D.7. Microservicios

Todo esto es sólo la estructura de la parte web y de control de sesión de la aplicación, ya que se dispone de la parte de minería la cual tiene su propia estructura de archivos...

Al ser tan simple como es, no se ha considerado necesario explicar todo lo que contiene. Los .py son *scripts* para ejecutar las funciones necesarias, la carpeta test tiene los test y el Dockerfile, como en el punto anterior, tiene los pasos para la construcción del contenedor.

Apéndice E

Documentación de usuario

E.1. Introducción

En este anexo mostramos en que sistemas de cliente se puede usar la aplicación.

E.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 entornos en los que ampliemos el número de servidores para mantener la sesión.

Navegadores:

1. Firefox 53
2. Chrome 57

E.3. Instalación

Al proporcionar nuestro servicio como una página web no necesitamos que el cliente o usuario instale la aplicación. De todas maneras la instalación de programador se puede usar de la misma manera, como en esa se recomienda *docker compose* ya que una vez copiado el código podemos ejecutar todo el sistema, aunque sea complejo con un simple comando.

E.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 a tener una aplicación con muchas

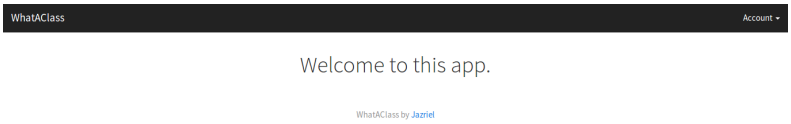

Figura E.1: Punto de entrada a la aplicación

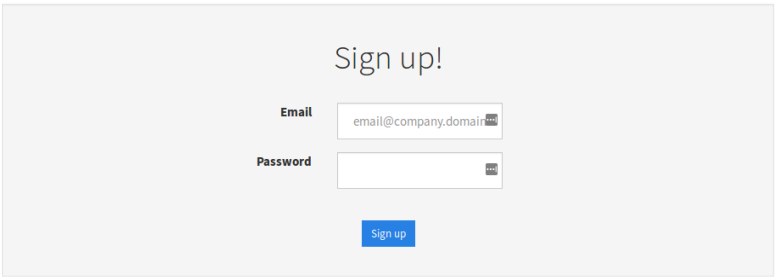

Figura E.2: Pantalla de registro de usuario

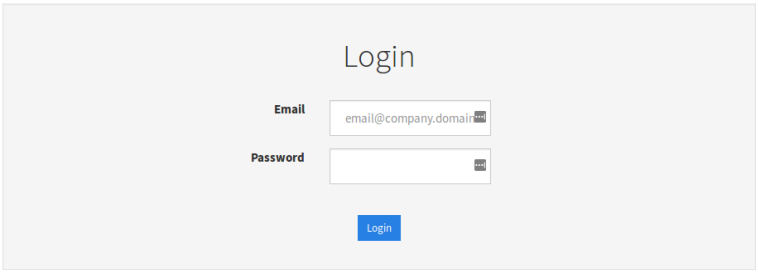

Figura E.3: Pantalla de inicio de sesión

funcionalidades sino a cómo hacerla.

El usuario generalmente llegará a la aplicación por su index, o índice.

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

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

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

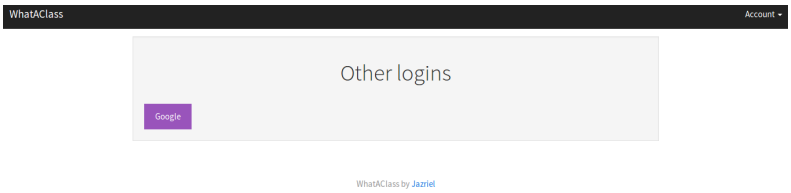

Figura E.4: Inicios de sesión alternativos

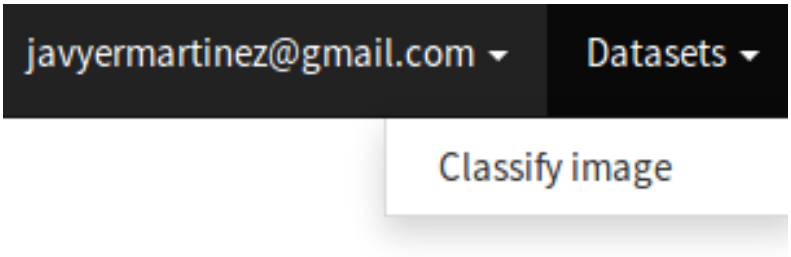

Figura E.5: Cambio de las zonas accesibles al iniciar sesión

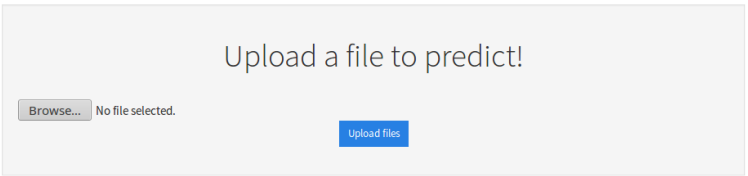

Figura E.6: Pantalla de acceso a el servicio de clasificación

Bibliografía

[1] License recommendations.

[2] Gnu lesser general public license, 1999.