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 — 28 de junio 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.

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

Tabla A.2: Costes de material al mes

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

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.

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)[**?**] , 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[**?**], 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.

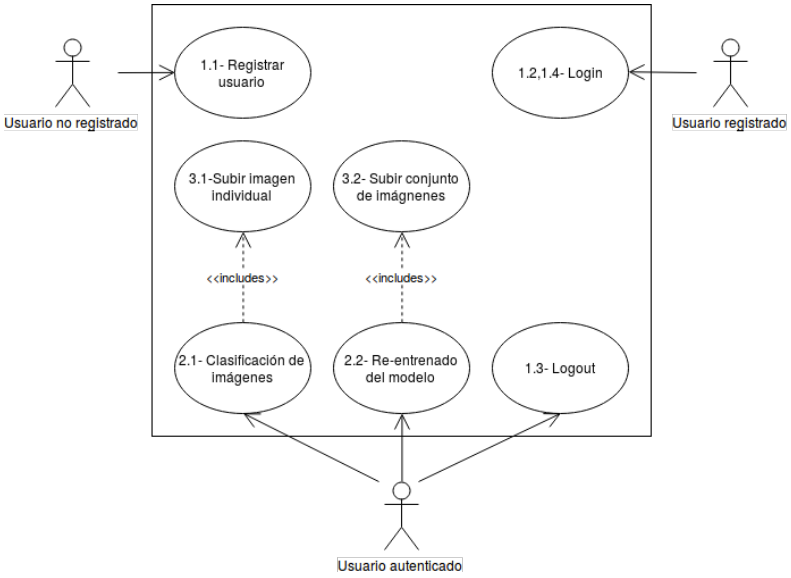

Figura B.1: Diagrama de casos de uso

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

C.2. Diseño de datos

C.3. Diseño procedimental

C.4. Diseño arquitectónico

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.

Para desplegar esta preparado para ser desplegado con docker cuya instalación se puede encontrar en el siguiente enlace: https://docs.docker.com/engine/installation/linux/ubun
install-using-the-repository

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

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

D.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 de el codigo podemos usar una herramienta como coverage y ejecutar test.py a traves de esa herramienta.

Apéndice E

Documentación de usuario

E.1. Introducción

E.2. Requisitos de usuarios

E.3. Instalación

E.4. Manual del usuario