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 — 29 de mayo de 2017
Tutor: nombre tutor

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

A.2. Planificación temporal

A.3. Estudio de viabilidad

Viabilidad económica

Viabilidad legal

Apéndice B

Especificación de Requisitos

B.1. Introducción

B.2. Objetivos generales

B.3. Catalogo de requisitos

B.4. Especificación de requisitos

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: documentación en código, que partes la tendran.
    * $create_db.py$: $scriptparacrearlabasededatosposiblementeseaeliminado.docker-compose.yml$: $dockercomposeparadespliegueconlabasededatosdirectamente$.

* 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

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

TODO: add apt install psycopg2 maybe something else too(?)

Para ejecutar en desarrollo lo mejor sería usar el script run.py 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.

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