UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Grado en Ingenieria Informatica

Generación automática de ficheros de configuración para el despliegue de servicios en Docker

Automatic generation of configuration files for deploying services in Docker.

Realizado por
Marcos Domínguez Moreno

Tutorizado por
José Miguel Horcas Aguilera

Departamento
Lenguajes y Ciencias de la Comunicación

MÁLAGA, Octubre de 2024

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADA/O EN TECNOLOGÍAS DE LA INFORMACIÓN

Generación automática de ficheros de configuración para el despliegue de servicios en Docker

Automatic generation of configuration files for deploying services in Docker

Realizado por
Marcos Domínguez Moreno

Tutorizado por
José Miguel Horcas Aguilera

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, OCTUBRE DE 2024

Fecha defensa: Diciembre de 2024

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Keywords: **A, B, C**

Resumen

Por revisar

Palabras clave: SPL, Docker , Despliegues Servicios , Desarrollo web, Integración Continua , Configuración , feature models , Linea de Productos

Índice

1 Introduccion

1.1. Motivación

Las herramientas de automatización de tareas son fundamentales para la productividad en el desarrollo de software, permiten mejorar la eficiencia, la calidad y la consistencia en la producción de código. Al relegar tareas repetitivas y propensas a errores a herramientas automatizadas, los desarrolladores pueden centrarse en tareas más creativas y de mayor valor añadido.

Parte del ciclo de vida en un desarrollo de software implica la configuración y el despliegue de servicios en entornos de producción y desarrollo. La configuracion y despliegue requiere de la creación de archivos de configuración específicos, como son por ejemplo los archivos dockerfile [1] y .dockerignore [2], que definen cómo se construye la imagen de un contenedor en tecnologicas como Docker [3]

En el contexto de la tecnología de contenedores, Docker se ha convertido en una de las plataformas más populares para la creación, el despliegue y la gestión de aplicaciones en entornos contenerizados. Sin embargo, la configuración de servicios en Docker puede ser una tarea compleja y propensa a errores, especialmente cuando se manejan múltiples servicios y configuraciones.

La motivación para realizar este proyecto surge de la necesidad de simplificar y automatizar el proceso de generación de archivos de configuración para el despliegue de servicios en Docker. En la actualidad, la creación manual de estos archivos dockerfile puede ser una tarea repetitiva y susceptible a fallos al no conocer el usuario de toda la informacion necesaria para la correcta configuracion de los servicios,

[1] dockerfile

[2] .dockerignore

[3] Docker

Al desarrollar una herramienta que automatice este proceso, se busca mejorar la eficiencia y la precisión en la configuración de entornos de contenedores, facilitando así el trabajo de desarrollo y administracion.

1.2. Objetivo

La meta principal de este proyecto es desarrollar una herramienta que permita la generación automática de archivos de configuración para el despliegue de servicios en Docker. Siguiendo las pautas recomendadas para la generacion de dichos archivos mediante de plantillas, las plantillas han de ser extensibles y adaptables a los cambios en un futuro sin necesidad de reescribir código.

Para ello se plantean los siguientes objetivos específicos:

- Investigar y analizar las tecnologías y herramientas existentes para la generación de archivos de configuración en Docker.
- Desarrollar una plantilla orientada a la generacion de ficheros dockerfile siguiendo las buenas practicas.
- Implementar un modelo de plantillas extensible y adaptable a los cambios en las tecnologías y las metodologías de desarrollo.
- Desarrollar una herramienta web que permita la selección de plantillas y la generación de archivos de configuración para el despliegue de servicios en Docker.
- Desplegar la aplicación en un entorno no orientado a producción y documentar el proceso de instalación y uso.
- Evaluar la eficacia y la usabilidad de la herramienta en la memoria.

1.3. Resultados Esperados

Dada la gran cantidad de opciones de configuración disponibles en para la generacion de ficheros de configuracion, el desarrollo de esta herramienta presenta varios desafíos técnicos y conceptuales. El primero de ellos es la magnitud de las opciones de configuración disponibles, que pueden variar en función de las necesidades y los requisitos de cada servicio. dar una covertura a todos los escenarios posibles es una tarea inabarcable, por lo que se ha optado por centrarse en los casos de uso más comunes y en las configuraciones más utilizadas en la práctica. Sin embargo se espera que la herramienta sea lo suficientemente flexible y extensible como para permitir la incorporación de nuevas plantillas y configuraciones en el futuro.

Los desarrollos de los servicios resultantes de la implementacion no tendran el objetivo de alcanzar un estado de produccion sino de servir como base para futuras implementaciones y desarrollos orientados a ello

1.4. Estructura del documento

- El Capitulo 1 describe la motivación del trabajo, sus objetivos, y los resultados esperados.
- El Capitulo 2 presenta una brebe introduccion a los contenidos, al estado del arte, soluciones existentes y que se aporta en este contexto.
- El Capitulo 3 describe las tecnologías y herramientas empleadas en el desarrollo del proyecto.
- El Capitulo 4 describe la metodología de trabajo empleada en el desarrollo del proyecto.
- El Capitulo 5 describe el modelado de las plantillas.
- El Capitulo 6 describe la arquitectura del proyecto y junto con la descripcion de los servicios
- El Capitulo 7 es un capitulo dedicado a la extensibilidad de las plantillas
- El Capitulo 8 describe el despliegue de la aplicación en un entorno no orientado a producción.
- El Capitulo 9 presenta los resultados obtenidos en el desarrollo del proyecto.

- El Capitulo 10 incluye las conclusiones del trabajo y las recomendaciones para trabajos futuros.
- El Apendice A Es un manual de Instalacion de la aplicacion
- El Apendice B Es un manual de Uso de la aplicacion
- El Apendice C Es una guia para el desarrollo de plantillas.

2

Estado del Arte

Para comprender el sentido de este proyecto y su problematica es necesario contextualizar algunos conceptos previos que son necesarios para comprender la situacion actual y contextualizar la problematica. A continuacion se introducen dichos conceptos de manera escalonada, abordaremos la problematica de la generacion de archivos de configuracion en Docker, las tecnologias empleadas en la generacion de dichos archivos y las soluciones actualmente disponibles.

2.1. Contenedores

Los contenedores son paquetes software ligeros que incluyen el código de las aplicaciones junto con sus dependencias, como versiones concretas de entornos de ejecución de ciertos lenguajes de programación y bibliotecas para ejecutar los servicios de software. [15]

2.1.1. Porque se usan los contenedores

Los contenedores constituyen un mecanismo de empaquetado lógico en el que las aplicaciones pueden extraerse del entorno en que realmente se ejecutan. Esta desvinculación facilita el despliegue uniforme de las aplicaciones basadas en ellos con independencia de que el entorno sea un centro de datos privado, la nube pública o el portátil personal de un desarrollador. [15] Es una solucion de facil aplicacion, alto rendimiento y extremadamente portable. [4]

[4]Existen restricciones en cuanto a la portabilidad de los contenedores, pero en general son facilmente desplegables en cualquier entorno enlace:https://stackoverflow.com/questions/42158596/can-windows-containers-be-hosted-on-linux

2.2. Imagenes

Las imágenes de contenedores son plantillas de solo lectura que contienen el código de la aplicación, las bibliotecas, las dependencias y otros archivos necesarios para ejecutar un contenedor. [1] No necesariamente necestiamos de una imagen para construir un contenedor [5] pero ese es el escenario más común. [6]

2.3. Docker

Docker [11] es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos Docker es una plataforma de código abierto que permite a los desarrolladores crear, desplegar y ejecutar aplicaciones en contenedores. [7]

2.3.1. Dockerfile

El archivo Dockerfile [9] es un archivo de texto que contiene una serie de instrucciones que Docker utilizará para construir una imagen de contenedor. Usualmente la construccion de un archivo de configuracion se realiza a partir de una imagen base, que puede ser una imagen oficial de Docker o una imagen personalizada creada por el usuario. A dicha imagen se añaden las instrucciones necesarias para instalar las dependencias y configurar el entorno de ejecución de la aplicación, lo que viene a representar el contenido de un archivo Dockerfile.

[5] Existe la posbilidad de crear contenedores de otras formas como a partir de otros contenedores

[6] contruir un contenedor a partir de una imagen lo hace reproducible

[7] Existen alternativas a Docker como Podman [28] pero Docker es la más popular y ampliamente utilizada en la actualidad.

2.3.2. Buenas Prácticas en la creacion de las imagenes

Para la creacion de los ficheros dockerfile es recomendable seguir una serie de buenas practicas para garantizar la eficiencia y la seguridad en el despliegue de servicios en Docker. Estas practicas se recojen en la documentacion oficial de Docker [10] En la documentacion se abordan una serie de recomendaciones para la creacion de imagenes de contenedores en Docker, entre las que se incluyen:

- Utilizar imágenes oficiales de Docker:
  - Las imágenes oficiales son mantenidas y actualizadas por la comunidad de Docker, lo que garantiza un alto nivel de seguridad y estabilidad.
  - Estas imágenes suelen estar optimizadas para un rendimiento eficiente y son ampliamente documentadas.
- Crea ficheros de configuracion dockerfile Multietapa:
  - Crear imágenes en varias etapas permiten dividir la construcción de la imagen, lo que puede mejorar el rendimiento
  - Esto ayuda a reducir el tamaño final de la imagen al incluir solo los archivos necesarios para la ejecución de la aplicación.
- Usar una imagen base ligera:
  - Utilizar imágenes base ligeras, como ‘alpine‘, puede reducir significativamente el tamaño de la imagen Docker.
  - Las imágenes ligeras también suelen tener menos vulnerabilidades de seguridad debido a su menor superficie de ataque.
- Reducir el número de capas:
  - Cada instrucción en un Dockerfile crea una nueva capa en la imagen. Reducir el número de capas puede mejorar el rendimiento y reducir el tamaño de la imagen.
  - Combinar múltiples comandos en una sola instrucción ‘RUN‘ puede ayudar a minimizar el número de capas.

- Eliminar archivos innecesarios:
    - Durante el proceso de construcción, es importante eliminar cualquier archivo temporal o innecesario para mantener la imagen lo más pequeña posible.
- Evitar el uso de la etiqueta ‘latest‘:
    - La etiqueta ‘latest‘ puede ser ambigua y llevar a problemas de consistencia y reproducibilidad.
    - Es mejor especificar una versión concreta de la imagen para asegurar que se está utilizando la versión correcta.
- Eliminar archivos temporales y de instalación:
    - Durante la instalación de paquetes y dependencias, se generan archivos temporales que deben ser eliminados para reducir el tamaño de la imagen.
    - Utilizar comandos como ‘apt-get clean‘ y ‘rm -rf /var/lib/apt/lists/*‘ para limpiar los archivos de instalación.

[8]

[8]Para más informacion: https://docs.docker.com/build/building/best-practices/

2.4. Problematica

La dificultad de crear imagenes reside en la naturaleza de la aplicacion que queremos llevar a un contenedor, existen una gran cantidad de escenarios donde si bien la tarea es sencilla, esta puede escoder una complejidad implicita de la que el usuario no es consciente. Estos apartados repercuten en el tiempo de compilación, tamaño de la imagen, mantenibilidad, seguridad y repetibilidad en las imagenes que se crean.
La calidad de las imagenes pueden marcan una diferencia que puede llegar a ser crítica para posteriores etapas en el desarrollo de aplicaciones como son la Orquestacion de contenedores [17].

2.5. Soluciones Actualmente Disponibles

En la actualidad hay disponibles varias soluciones para disminuir la complejidad implicita que reside en en la configuracion de dichos ficheros dockerfile. Existen más herramientas que las representadas a continuacion pero estas son las más empleadas.

2.5.1. Docker init

Docker init es el comando de Docker Desktop [7] que permite inicializar un proyecto con un archivo de configuración básico. su funcionamiento consta de elegir una plantilla de proyecto y seleccionar entre las opciones de configuracion. Las opciones de configuracion se limitan a elegir el lenguaje de programacion, puerto en el que despliega la aplicacion y que comando se ejecutara al iniciar el contenedor. Las plantillas son creadas y mantenidas por la comunidad [9]

[9] En este proyecto se reutilizan y se recogen ideas de varias de las plantillas que incluye Docker Init

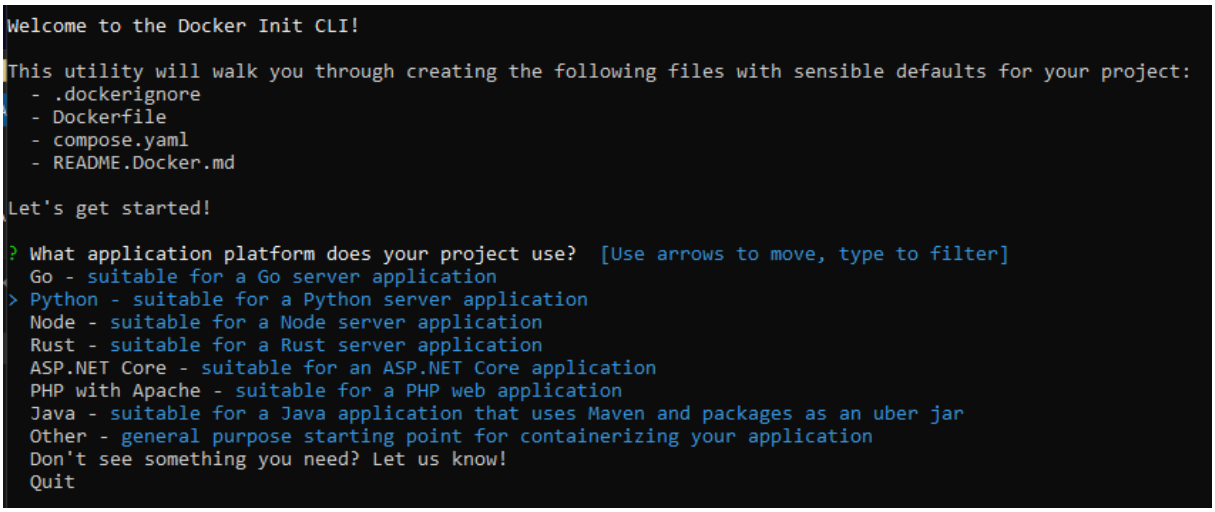

Figura 1: Captura de las opciones de Plantillas disponibles Docker Init.

2.5.2. Docker Extension for Visual Studio Code

La extension de Docker para visual estudio code [5] permite entre otras muchas de sus funcionalidades la generacion del fichero dockerfile entre otros. su comportamineto es similar al de Docker init, permitiendo entre otras opciones elegir el lenguaje de desarrollo del proyecto y que comando se ejecutara al iniciar el contenedor.

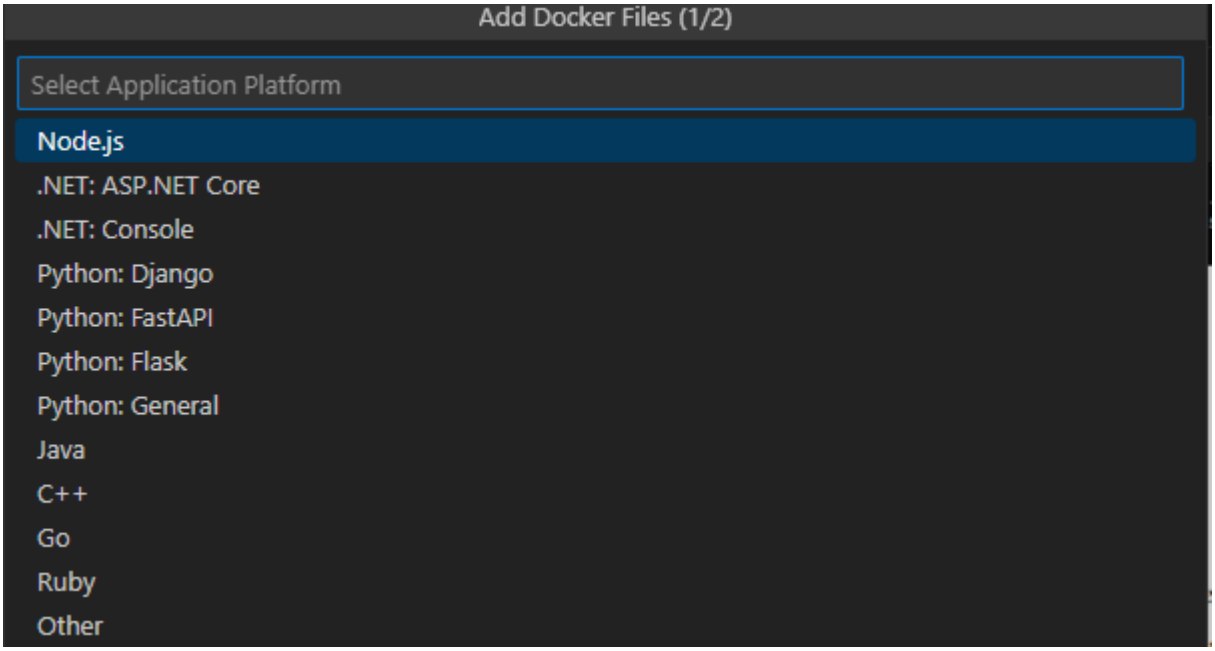

Figura 2: Plantillas Disponibles en la Extesnsion para Docker de Visual Studio Code.

```dockerfile
# For more information, please refer to https://aka.ms/vscode-docker-python
FROM python:3-slim

EXPOSE 5002

# Keeps Python from generating .pyc files in the container
ENV PYTHONDONTWRITEBYTECODE=1

# Turns off buffering for easier container logging
ENV PYTHONUNBUFFERED=1

# Install pip requirements
COPY requirements.txt .
RUN python -m pip install -r requirements.txt

WORKDIR /app
COPY . /app

# Creates a non-root user with an explicit UID and adds permission to access the /app folder
# For more info, please refer to https://aka.ms/vscode-docker-python-configure-containers
RUN adduser -u 5678 --disabled-password --gecos "" appuser && chown -R appuser /app
USER appuser

# During debugging, this entry point will be overridden. For more information, please refer
CMD ["gunicorn", "--bind", "0.0.0.0:5002", "myapp.py:app"]
```

Figura 3: Captura de una plantilla generada por la Extesion para Docker de Visual Studio Code

2.5.3. Inteligencia Artificial

En la actualidad existen soluciones que se apoyan en la inteligacia Artificial para la generacion de archivos de configuracion, y para la creacion de los fichero dockerfile. La calidad de estas soluciones es variable y depende de la calidad de los modelos de lenguaje natural empleados y de la calidad del prompt[10] de entrada. como magickpen [19] [11] o alternativas con propositito más general como chatgpt [27] [12]

[10]Instrucción o texto inicial proporcionado a una herramienta generativa de IA para dirigir la generación de respuestas o resultados específicos [18]

[11]magickpen:https://magickpen.com/templates/128/

[12]chatgpt:https://chat.openai.com/

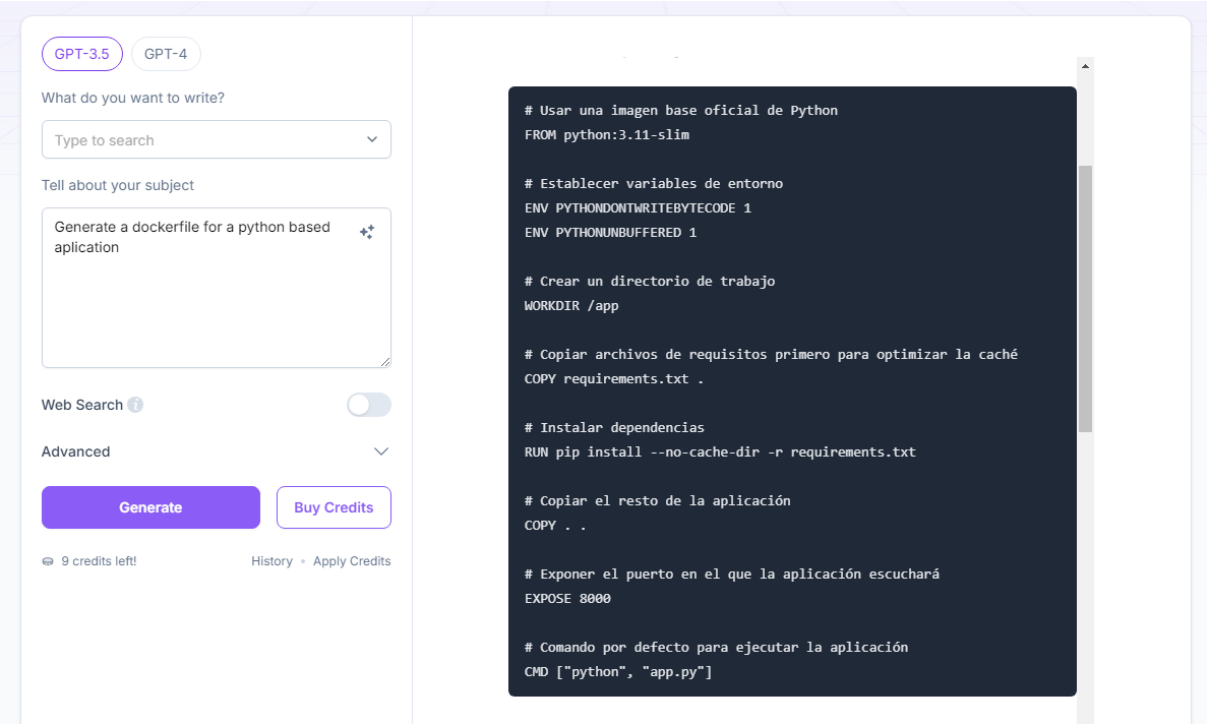

Figura 4: Captura de un resultado de magickpen

2.6. Conclusiones Preliminares

En este estudio se han recogido algunas de las principales herramientas y tecnologías disponibles en la actualidad para la generación de archivos de configuración. Todas las herraminetas (Salvo propiamente los Modelos de Texto Generativo) hacen uso de plantillas para la generacion de los ficheros dockerfile y dockerignore, estas plantillas si bien son correctas son por naturaleza limitadas y no permiten la personalizacion más final de las opciones de configuracion. No obstante suponen un punto de partida para el desarrollo de aplicaiones

2.7. Que se aporta en este contexto:

Las opciones actualemtne disponibles para la generacion de archivos de configuracion para el despliegue de servicios en Docker son limitadas y no permiten la personalizacion de las opciones de configuracion. Se deja de lado la variabilidad y las posibles opciones que existen a la hora de construir las imagenes, el proyecto proporcinara una nueva forma de construir plantillas dotando capacidad al usuario de una personalizacion más profunda y detallada de las opciones de configuracion de los servicios en Docker.

2.7.1. Software Product Line SPL

El proyecto propone una solucion en linea con la implementacion de plantillas pero enfocando estas plantillas como si formaran parte de una linea de productos software. Una línea de productos software (SPL del inglés Software Product Line) es una familia de productos software relacionados entre sí que comparten ciertas características comunes (similitudes) pero que también tienen características variables. CITA Documentacion

En un SPL, los sistemas se descomponen en características (features) que representan funcionalidades o comportamientos del sistema. El principal artefacto para modelar la variabilidad en SPL son los modelos de variabilidad. Existen muchos modelos de variabilidad pero los más extendidos y usados en la práctica son los modelos de características (feature models).

Un feature model (FM) [34] es un modelo para representar la variabilidad de una SPL en base a características comunes y variables (véase Figura 1), especificando qué características se pueden seleccionar en una configuración para generar un producto válido de la SPL. CITA

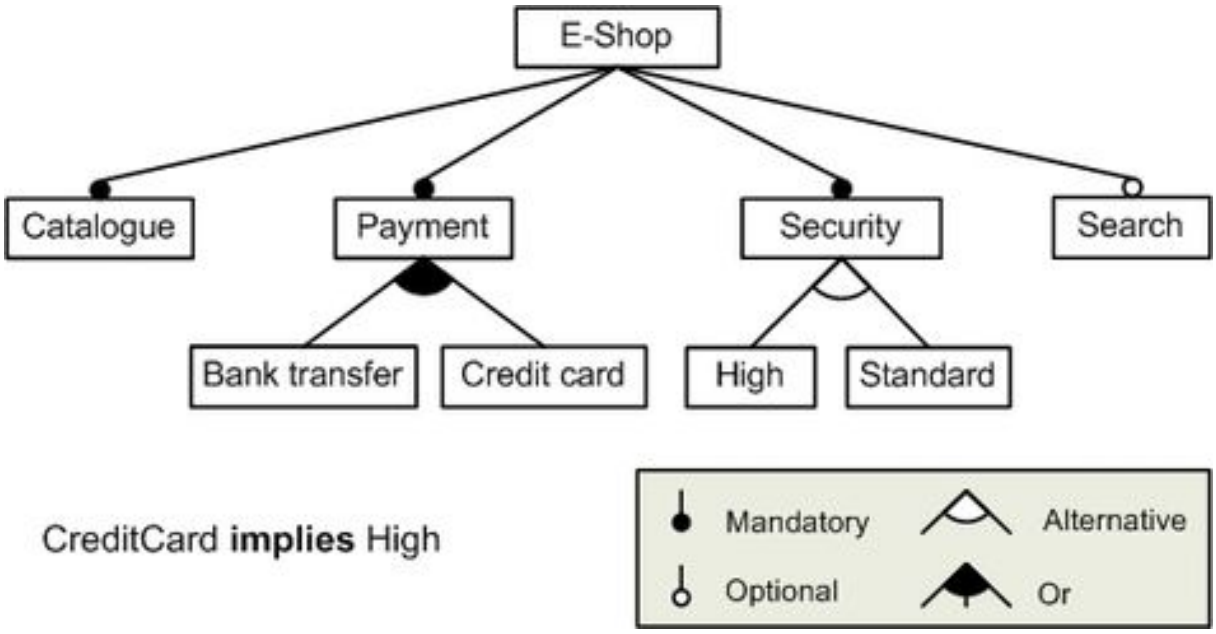

Figura 5: Ejemplo Represantcion Grafia de un feature model [34]

3

Tecnologías Empleadas

Este apartado describe las tecnologías y herramientas empleadas más relevantes para el desarrollo del proyecto, así como su función y su relación con el objetivo del proyecto.

3.1. Universal Variability Language UVL

La Universal Variability Language (UVL) [6] es un lenguaje de modelado diseñado específicamente para representar y manejar la variabilidad en sistemas software. La variabilidad en software se refiere a las partes de un sistema que pueden personalizarse, configurarse o cambiarse para adaptarse a diferentes necesidades o contextos. UVL se usa comúnmente en el contexto de ingeniería de líneas de productos de software (Software Product Line Engineering o SPLE), donde se trabaja con una familia de productos que comparten un núcleo común, pero pueden tener variaciones. todas las plantillas que se han desarrollado en este proyecto requieren de un archivo UVL para poder generar configuraciones validas.

3.2. Docker

La apliccion web tanto como todos sus servicios han sido desarrollados con el objetivo en mente de ser ejecutados dentro de contenedores Para el manejo de los contenedores en el entorno de desarrollo se ha optado por la utilizacion de Docker Desktop es complemento con la consola de comandos. Esta herremienta ayuda al proyecto en el momento en que lo hacer portable y facil de desplegar en cualquier entorno.

3.3. Node.js

[24] para el desarrollo de la aplicacion web se ha optado por la utilizacion de Node.js como entorno de ejecucion de JavaScript para el desarrollo Frontend. Esta herramienta constituye una plataforma de desarrollo en tiempo de ejecución cuenta con la ventaja de contar con una extensa biblioteca de módulos que simplifican el desarrollo de aplicaciones web. Con npm (Node Package Manager) [26], Node.js ofrece acceso a una vasta colección de paquetes y módulos que pueden ser fácilmente integrados en el proyecto, acelerando el desarrollo y reduciendo la necesidad de escribir código desde cero. Para el proyecto se emplea la version v22.10.0 y la version v10.9.0 del gestor de paquetes npm

3.4. Angular

[2] En cuanto al desarrollo de la aplicacion web se ha optado por la utilizacion de Angular como framework de desarrollo. El motivo de esta decision es la idea de emplear un framework moderno para emplear los metodos de desarrollo actuales y las mejores practicas en el desarrollo de aplicaciones web. Emplear otros framework de desarrollo como React o Vue.js tambien hubiera sido una opcion valida al existir una cierta convergencia entre ellos en cuanto a funcionalidades y caracteristicas. Para los requisitos del proyecto entendimos que Angular ya integra gran parte de las funcionalidades que requesia nuestra aplicacion ahorrando tiempo en la implementacion de las mismas. La version empleada es Angular v18 [13]

3.4.1. Angular Material

[3] Para el diseño de la interfaz grafica de la aplicacion web se ha optado por la utilizacion los componentes provenientes de los modulos de Angular Material. Siguiendo la linea de diseño de Material Design, Angular Material proporciona una serie de componentes y directivas que facilitan la creacion de interfaces de usuario modernas y atractivas. Los componentes tienen una reaccion conocidad por el usuario y son faciles de implementar en la aplicacion web. [14]

[13] Angular 18 no es la ultima version disponible pero al depender de otros desarrollo se ha optado por la utilizacion de esta version

[14] La ultima version disponible es 18.2.9

3.4.2. Monaco Editor

[21] Para la edicion de los archivos de configuracion se ha optado por la utilizacion de Monaco Editor como editor de codigo. Monaco Editor es un editor de codigo fuente basado en la web que se utiliza en varios proyectos de Microsoft, como Visual Studio Code, Azure DevOps y el editor de codigo de GitHub. cuenta con una serie de caracteristicas que lo hacen ideal para la edicion de archivos de configuracion, como la resaltado de sintaxis, el autocompletado y la verificacion de errores. tambien es altamente personalizable y extensible, lo que permite adaptarlo a las necesidades del proyecto. [15]

3.4.3. ngx-json-viewer

[25] Para la visualizacion de los archivos de configuracion se ha optado por la utilizacion de ngx-json-viewer como visor de archivos json. ngx-json-viewer es un componente de Angular que permite visualizar archivos JSON en un formato legible y estructurado. es un proyecto de codigo abierto que se puede integrar facilmente en aplicaciones Angular y personalizar segun las necesidades del proyecto.

3.5. Nginx

[23] Nginx es un servidor web de código abierto desarrollado en C que se utiliza para servir contenido web estático y dinámico, así como para actuar como proxy inverso y balanceador de carga. la ventaja de Nginx es principalmente la poca necesidad de configuracion que requiere para su correcto funcionamiento, ademas de ser un servidor web ligero y de alto rendimiento. Para las necesidades del proyecto se ha empleado en sus funciones de proxy inverso para redirigir las peticiones de la aplicacion web al servidor de backend. y como servidor web para servir el contenido estatico de la aplicacion web.

3.6. Python

[12] Python es un lenguaje de programación de alto nivel, interpretado y de propósito general que se ha convertido en uno de los lenguajes más populares en la actualidad. Para el

[15] La ultima version de angular en la que es compatible es la v18 por este motivo se ha desarrollado en esta version de Angular https://www.npmjs.com/package/ngx-monaco-editor-v2

desarrollo de los servicios de backend se ha optado por la utilizacion de Python como lenguaje de programacion. Python es conocido por su sintaxis clara y legible, su amplia biblioteca estándar y su soporte para múltiples paradigmas de programación, como la programación orientada a objetos, la programación funcional y la programación imperativa.

3.6.1. Flask

[29] Flask es un microframework de Python para el desarrollo de aplicaciones web. Flask es conocido por su simplicidad y su facilidad de uso, lo que lo convierte en una excelente opción para el desarrollo de aplicaciones web pequeñas y medianas.

3.6.2. gunicorn

[16] Gunicorn es un servidor HTTP WSGI para Python que se utiliza para servir aplicaciones web desarrolladas en Python. su objetivo es proporcionar un servidor web simple y eficiente que pueda manejar múltiples solicitudes simultáneamente. esta herraminta es necesaria para el despliegue de la aplicacion en un entorno de produccion.

3.6.3. Pypi

[31] El uso de Pypi como repositorio de paquetes de Python es necesario para la instalacion de las dependencias de la aplicacion. asi como ha sido empleado para subir paquetes diseñados para la aplicacion.

3.7. Jinja

[30] Jinja es un motor de plantillas para Python que se utiliza para generar archivos de texto dinámicos basados en plantillas.

3.7.1. Jinja2

Jinja2 es la versión más reciente de Jinja y se utiliza comúnmente en aplicaciones web de Python para generar contenido dinámico. Jinja2 es conocido por su sintaxis simple y legible, su capacidad para generar contenido dinámico y su integración con Python y otros marcos de desarrollo web. Jinja2 permite la generacion de archivos de texto dinamicos basados en

plantillas, lo que facilita la creacion de archivos de configuracion personalizados y adaptables a las necesidades del usuario. Todas las plantillas desarrolladas han usando la sintaxis de Jinja2

3.8. Git

[32] Para el control de versiones se ha optado por la utilizacion de Git como sistema de control de versiones.

3.9. GitHub

[13] GitHub es una plataforma de desarrollo colaborativo que se utiliza para alojar proyectos de código, realizar seguimiento de problemas y tareas, y colaborar con otros desarrolladores. Todo el proyecto es alojado en GitHub incluyendo plantillas, codigo ...

3.9.1. GitHub Public Api

[14] Para la gestion de las plantillas se ha empleado la API publica de GitHub para la obtencion de las plantillas disponibles. Esta desion se ha tomado por facilidad de uso, ya que la API de GitHub es ampliamente conocida y documentada, y permite acceder a una gran cantidad de información sobre los repositorios y las plantillas disponibles. Esto supone que las plantillas estan visibles y son accesibles y modificables por cualquier usuario de GitHub. Simplificando el proceso de creacion, mantenimiento y actualizacion de las plantillas. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

3.10. Visual Studio Code

[22] Visual Estudio Code es un editor de código fuente desarrollado por Microsoft que se utiliza para escribir, editar y depurar código. Su mencion se debe a que es el editor de codigo fuente que se ha empleado para el desarrollo del proyecto. Ademas de ser clave en el proceso de generador de ficheros de configuracion de los que la aplicacion hace uso.

3.10.1. UVLS extension

[4] UVLS es una extension de Visual Studio Code que proporciona soporte para la Universal Variability Language (UVL) en el editor de codigo. La extension UVLS proporciona resaltado de sintaxis, autocompletado, verificacion de errores y otras funcionalidades que facilitan la edicion de archivos de configuracion en UVL. ademas la clave más importante es que es la unica herramienta existente que permite crear una configuracion correcta partiendo desde un Feature Model

Otras extensiones de menos importancia empleadas en *VSCode* son:

- **Docker Extesnion**
- **Flama**
- **Graphviz Interactive Preview**
- **Latex Workshop**
- **Pylance**
- **Python Debugger**
- **Better Jinja**

4

Metodologia del trabajo

4.1. Introduccion

El enfoque se ha dado para abordar el desarrollo de este trabajo ha consistido en combinar un trabajo previo de analisis e investigacion con un desarrollo iterativo y agil. La investigacion y analisis tenia el objetivo de conocer en profundidad las tecnologias y herramientas empleadas para porstesiromete estudiar una posible aplicacion de estas ideas y conceptos en el desarrollo de las plantillas. una vez resuelta esta primera parte se ha procedido a una toma de requisitos y a la planificacion de la implementacion de la aplicacion. para la implementacion de la aplicacion se ha optado por la utilizacion de una metodologia de desarrollo agil que permita la adaptacion a los cambios y la evolucion del proyecto.

4.2. Metodología del Desarrollo

Al encontrarnos ante un trabajo individual toda la responsabilidad y capacidad del desarrollo recae sobre una persona. Para el desarrollo de código optaremos por una metodología Scrum simplificada. De forma iterativa se ejecutará un Sprint con unos objetivos de desarrollo y planificación muy limitados. Con el objetivo de dar cabida a la complejidad del proyecto, con animo de encajar nuevas ideas y revisiones que pudieran sucederse. Tras cada Sprint se evaluará la progresión y dirección del proyecto con el tutor.

4.2.1. Justificación

El desarrollo de las plantillas requiere un estudio con el fin de limitar las opciones de configuración y centrarse en los casos de uso más comunes y en las configuraciones más utilizadas

en la práctica. De entrada aportar una solucion tecnica optima a un problema es extremadamente improbable, por lo que se opta por una metodologia de desarrollo agil que permita la adaptacion a los cambios y la evolucion del proyecto.

4.2.2. Procedimiento

Tras cada Sprint se evaluará la progresión y dirección del proyecto con el tutor en el escenario de una reunión de revisión.

4.2.3. Limitaciones

El desarrollo de un proyecto de estas características requiere de una planificación y una metodología de trabajo adecuada. En este sentido, como se han identificado anteriormente una serie de limitaciones y restricciones que pueden afectar al desarrollo del proyecto. para el caso el desarrollo de las plantillas tiene que verse limitado en cuanto a formato y contenido, ya que no se puede abarcar todas las posibles opciones de configuracion.

4.3. Fases de Estudio

4.4. Estudio del Estado del Arte :

En esta fase se realiza una revisión exhaustiva de la literatura y tecnologías existentes relacionadas con el tema del proyecto. El objetivo es comprender el panorama actual, identificar tendencias, mejores prácticas y tecnologías relevantes que puedan influir en el desarrollo del proyecto. Los puntos clave de esta fase son:

- Para comprender la tecnologia de los contenedores fue exencial la lectura de Modern Infrastructures and Applications with Docker [20]
- Completar la intruduccion al lenguaje uvl con UVL Playground [33]

4.5. Investigación dockefiles

Esta fase implica investigar y evaluar las diferentes opciones de configuracion que se presentan en el desarrollo de un fichero dockerfile en el despliegue de aplicaciones. Se exploran las

características, herramientas y técnicas que pueden optimizar el uso de Docker en un proyecto software.

- Para compreder la importacia de las buenas practicas en la creacion de imagenes fue necesario la lectura de la documentacion oficial de Docker Dockerfile reference [8]

Es necesario conocer como se aplica a la practica la creacion de imagenes. para ello han estudiado entre otros los siguientes proyectos:

4.6. Desarrollo de Modelos de plantillas

Esta fase implica la construcción el árbol de características y las restricciones textuales que orienten el espacio de configuraciones posibles del fichero Dockerfile. En este paso se pondran en practica todo el conocimiento adquirido en los pasos anteriores que tendran como resultado el modelado de unas plantillas. Hemos de tener en cuenta las buenas practicas y las recomendaciones de la comunidad para la creacion de los ficheros dockerfile CITA. Estas practicas tienes que estar presentes en el diseño de las plantillas desde el inicio, tambien hemos de generar test para mantener la consistencia de las implementaciones. Este paso queda cubierto con más detalle en el Capitulo 5

4.7. Fase Implementacion Scrum

4.7.1. Requisitios de la aplicación

Aquí se definen y documentan los requisitos funcionales y no funcionales de la aplicación. Se recopilan y analizan las necesidades de los posibles usos, los objetivos del proyecto y las restricciones técnicas para establecer una planificacon del desarrollo. Los requisitos han cambiado conforme el proyecto avanzaba en sus etapas de desarrollo sin animo de ser exhaustivos las funcionalidades mínimas que se pretenden alcazar son en cuanto a la generacion de productos

La aplicacion web debe permitir seleccionar la plantilla y la version con la que se desea trabajar Las plantillas pueden recargarse en del repositorio La aplicacion web debe generar un el producto en el momento en el que se aplique una configuracion valida

4.7.2. Desarrollo del la aplicación de uvengine

En esta fase se desarrolla la aplicación de uvengine, que permite la generación automática de archivos de configuración

4.7.3. Desarrollo de la web

En esta fase se desarrolla la aplicación web que permite la interacción con el usuario y la visualización de los archivos de configuración generados.

4.7.4. Desarrollo de los servicios de backend

En esta fase se desarrollan los servicios de backend que permiten la comunicación entre la aplicación web y la aplicación de uvengine y los repositorios.

4.7.5. Elaboración de la guía de uso

En esta fase se elabora una guía de uso que explica cómo utilizar la aplicación y las funcionalidades disponibles. Se elabora una guía detallada que describe cómo utilizar la aplicación, incluyendo instrucciones paso a paso, capturas de pantalla y ejemplos.

4.7.6. Elaboración de un manual de instalación

En esta fase se elabora un manual de instalación que explica cómo instalar y configurar la aplicación en un entorno local.

4.8. Elaboración de la memoria

En esta fase se redacta la memoria o informe final del proyecto, que documenta todo el proceso de desarrollo, desde la planificación hasta la implementación y las lecciones aprendidas. La memoria incluirá el análisis de los resultados, conclusiones y recomendaciones para trabajos futuros.

5

Modelado de las plantillas

Este espacio pretende describir de forma detallada el modelado de las plantillas y a que necesidad responden en la contruccion de un fichero de configuracion

5.1. Introducción

Dar cabida a la gran cantidad escenarios y posibles caso de uso para la apliacion se ha descrito anteriormente como una tarea abrumadora. Aunque pudieramos crear una plantilla lo suficientemente expresiva para dar cabida a todos los escenarios de uso exitentes hasta la fecha, las plantillas quedarian desfasadas conforme se suceden cambios en las metodologias, surgen nuevas imagenes base, otras quedan desactualizadas etc. La obra resultante llegaria a ser inpracticable en cuanto a la cantidad de opciones y configuraciones que se podrian dar. El objetivo del desarrollo todas las plantillas de este proyecto no pretende llegar a un nivel extremadamente avanzado de personalizacion, sino que sirvan como base para la creacion de plantillas más especificas y detalladas.

5.2. Componentes de las plantillas

Las plantillas se dividen en varios componentes

5.2.1. Featuremodel(.uvl)

El archivo uvl es el archivo principal de la plantilla, contiene la descripcion de las caracteristicas y las restricciones de la plantilla. en este archivo dicta principalmente como se generera una configuracion valida que sea capaz de cumplir con las restricciones textuales IMAGEN

CITA

5.2.2. Plantillas Jinja

Las plantillas Jinja son los archivos de plantilla que se utilizan para generar los productos a partir de una configuracion valida o no del modelo. CITA La idea es exponer estas plantillas ante una configuracion ya validadada con herramientas como UVLS CITA, Estas plantillas constituyen el nuecleo del funcionamiento, en ella se declaran de forma explicita las opciones de configuracion y las variables que se pueden modificar. Las plantillas tienen que tener la esctrutura necesaria como para Usando la Sintaxis de Jinja crear un producto valido para una configuracion existente

5.2.3. Mapping Model

Es el unico componente completamente Opcional IMAGEN Contiene la extructura de un fichero csv de tres con columnas. Las primeras columnas permiten mapear features sean estas booleanas o no a otros feature Boolenas que podran ser empleadas en las plantillas jinja, la tercera columna permite especificar una feature Este archivo permite renombrar features y hacerlas más comprensibles para las plantillas jinja. Tambien es el responsable de hacer funciar configuraciones sobre otras plantillas.

5.3. Plantilla dockerfile

Como hemos visto un fichero Dockefile es un archivo de texto que contiene una serie de instrucciones que Docker CITA utilizará para construir una imagen de contenedor. la plantilla para dockerfile cuenta con el modelo más grande del proyecto, La idea para crear estos modelso es subdividir el modelo en partes más pequeñas y manejables, de forma que se pueda trabajar con ellas de forma independiente. El criterio a la hora de hacer subdivisiones es similar al criterio que usan repositorios como Docker HUB CITA donde divide sus imagenes por categorias dependiendo de su funcion. En este contexto podemos construir una primera subdivision que represente el resultado final esperado, es decir la primera feature sera decidir si nuestra imagen tiene el objetivo de desplegar un contenedor que albergue un Frontend, Backend, Base de datos, etc.

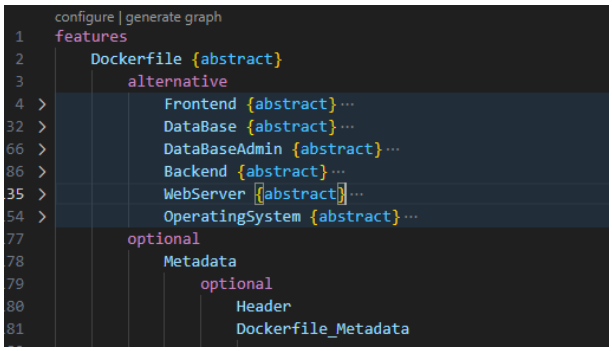

Figura 6: Captura de la primera subdivision para dockerfile.

Cada una de estas categorias contendra a su vez una serie de subcategorias que representen las opciones de configuracion que se pueden dar en cada una de ellas. Centremonos en el caso de uso de un Frontend, Comunmente al deplegar una imagen de frontend se sigue una estrategia de multi etapa CITA donde en una primera etapa compilamos la aplicacion y en una posterior etapa la alojamos en un servidor web, esto con el objetivo de crear una imagen lo más pequeña posible al no contener ninguno de los ficheros fuente

en este caso las subcategorias podrian ser el lenguaje, el puerto en el que se desplegara la aplicacion, el comando que se ejecutara al iniciar el contenedor, etc. Estos son elementos ya mecionados en la seccion de herramientas disponibles como pueden ser Docker Init CITA, Nuestro modelo permite recojer estas caracteristicas además de otras como pueden ser el servidor web que queremos junto con su version, Podremos seleccionar el framework de desarrollo que se ha empleado, la version en la que compilara la aplicacion etc. Las posibilidades van mucho más alla de lo que se recoje en esta version del fichero dockerfile pero estas opciones ya expanden enormemente la capacidad de configuracion que tienen otras herremientas.

5.3.1. Plantillas Jinja2

Una vez que contamos con el modelo de la plantilla, podemos proceder a la creacion de las plantillas Jinja2 que son quienes realmente aportan el contenido al producto. La sintaxis de Jinja2 nos permite aplicar cierta lógica a la hora de generar los ficheros de configuracion, por ejemplo podemos emplear condicionales para que ciertas partes del fichero se generen o no en funcion de la configuracion que se haya dado. Dividir las plantillas no solo es util porque las hace más manejables y faciles de mantener, sino que tambien permite la reutilizacion de

Figura 7: Captura de la primera subdivision para dockerfile.

las mismas, por ejemplo si tenemos una plantilla que se repite en varias plantillas, podemos extraerla a un fichero aparte y reutilizarla en todas las plantillas que lo necesiten. Para este caso podemos generar una plantilla para los metadatos y labels de la imagen

5.3.2. Test sobre la Plantilla dockerfile

El diseño y construccion de las plantillas es un proceso incremental, debe comprobarse con regularidad si nuestra intencion a la hora de seguir desarrollar una plantilla es mantener una coherencia con lo añadido hasta el momento. Con esta idea en mente la solucion propuesta consiste en la creacion de pruebas que validen la correcta construccion de productos segun la plantilla en la que estamos trabajado. Estas pruebas deben ser capaces de comprobar que las plantillas generan un producto correcto. para el caso del dockerfile estas pruebas deben componerse por cada configuracion y un producto esperado, si nuetra plantilla es capaz dada lo configuracion de entrada generar el mismo producto la prueba sera considerada como exitosa.

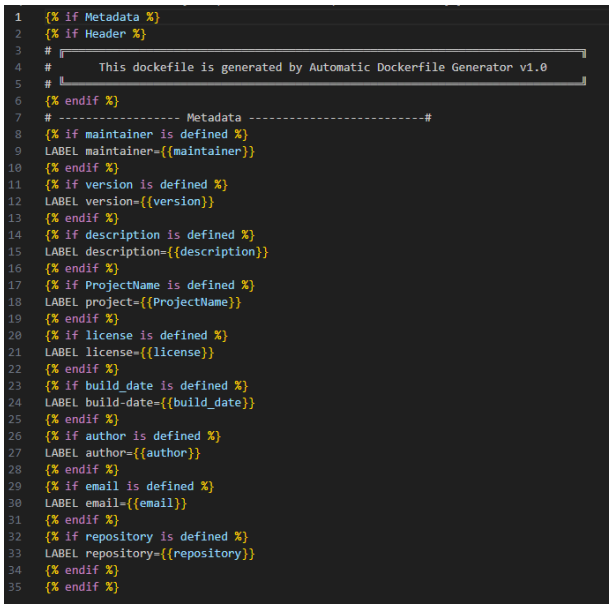

Figura 8: Captura de la primera subdivision para dockerfile.

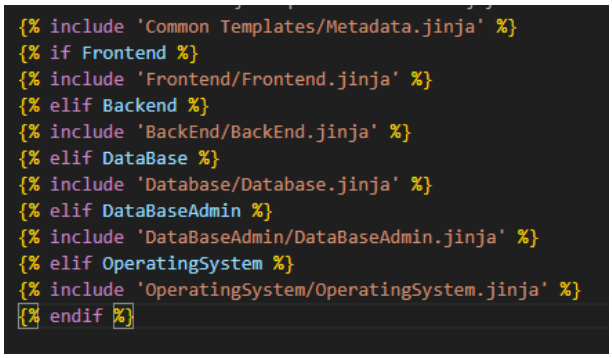

Figura 9: Captura de la primera subdivision para dockerfile.

es evidente que ciertos cambios más radicales plantean refactorizar algunos test. aun asi es aconsejable validar el diseño de los nuevos elementos de la plantilla solucionando estas incoherencias. En el contexto de esta plantilla dockerfile las pruebas se han realizado empleado las librerias de pytest CITA para la generacion de los productos esperados. para cada una de las primeras divisiones de las plantillas se generan ficheros de test para clasificar las configuraciones que se prueban y los productos esperados.

```
# ------------------ Build Stage ----------------------#

{% include 'Frontend/Framework/Framework.jinja'%}

# ------------------ Production Stage ------------------#

{% include 'Frontend/WebServer/WebServer.jinja'%}

{%- if Port is defined %}

EXPOSE {{Port}}
{% else %}

EXPOSE 80
{% endif %}

```

Figura 10: Captura de la primera subdivision para dockerfile.

```
{
  "file": "Dockerfile_fm.uvl",
  "config": {
    "Backend": true,
    "Programing_Language": true,
    "Python": true,
    "Flask": true,
    "Install_Python_Requirements": true,
    "WSGI": true,
    "Gunicorn": true,
    "interface_Gunicorn": "0.0.0.0",
    "workers_Gunicorn": "4",
    "module_Gunicorn": "holamundo.py",
    "callable_Gunicorn": "app",
    "Gunicorn_Install": true
  }
}
```

Figura 11: Conguracion Ejemplo 1 Backend.

```dockerfile
# Python Aplication Dockerfile
# Fask Aplication Dockerfile

# Avoid using latest tag in production
FROM python:latest

# Keeps Python from buffering stdout and stderr to avoid situations where
# the application crashes without emitting any logs due to buffering.
ENV PYTHONUNBUFFERED=1

# Create a non-privileged user that the app will run under.
# See https://docs.docker.com/go/dockerfile-user-best-practices/
ARG UID=10001
RUN adduser \
    --disabled-password \
    --gecos "" \
    --home "/nonexistent" \
    --shell "/sbin/nologin" \
    --no-create-home \
    --uid "${UID}" \
    appuser
# Install the application dependencies
RUN --mount=type=cache,target=/root/.cache/pip \
    python -m pip install --no-cache-dir -r requirements.txt

WORKDIR /app

COPY . /app
# Install Gunicorn This is not recommended for production, requirements.txt should be used instead

RUN pip install --no-cache-dir gunicorn

USER appuser

# Run the application using Gunicorn
CMD ["gunicorn", "-w", "4", "-b", "0.0.0.0:80", "holamundo.py:app"]

# Make port 5000 available to the world outside this container
EXPOSE 5000

```

Figura 12: Producto Esperado.

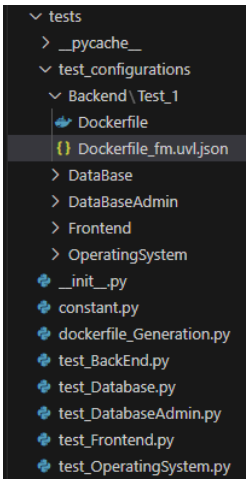

Figura 13: Captura de la primera subdivision para dockerfile.

6

Arquitectura y Descripcion del sistema

Esta seccion describe la arquitectura y el modelado del sistema, incluyendo los componentes, la estructura y el diseño de la aplicación resultante.

6.1. Vision general de la arquitectura

La arquitectura de la aplicación se divide en dos partes principales: el backend y el frontend. El backend es responsable de la generación de los archivos de configuración a partir de las plantillas y de la comunicación con los repositorios de plantillas. El frontend es responsable de la interacción con el usuario y la visualización de los archivos de configuración generados.

6.2. Descripcion de servicios de backend

Para los servicios de backend estos se han dividido en varios servicios que se encargan de tareas especificas. La primera de ellos es Repository Manager CITA este servicio se encarga de servir traducir las peticiones que puede realizar el front end a la API de GitHub. Estas peticiones son descargar el archivo uvl de la plantilla y version seleccionada, descargar el listado de plantillas disponibles y descargar el listado de versiones disponibles para cada plantilla

6.2.1. Repository Manager

Esta aplicaicon esta desarrollada en Python y emplea el framework Flask para la creacion de la API, esta API se encarga de servir los archivos uvl de las plantillas y las versiones dispo-

nibles. para hacer funcionar esta API se emplea la api publica de github que permite acceder al contenido de repositorios. En este caso el repositorio al que queremos acceder es el repositorio Templates CITA que contiene las plantillas disponibles.

6.2.2. UVEngine Resolver

La segunda de ellas es UV Engine Resolver CITA este servicio se encarga de resolver la variabilidad a partir de recibir como parametros Un fichero Json que contiene el nombre de la plantilla, la version a utilizar y la configuracion el resultado de la funcion sera un producto

6.3. Descripcion de Frontend

7 Extensibilidad

En esta seccion se pretende hacer incape en la extensibilidad de la aplicacion, como se ha mencionado anteriormente el proyecto tiene en mente que pueda adaptarse a las necesidades futuras.

Al almacernar las plantillas en un repositorio publico de GitHub, se permite a cualquier usuario modificar, añadir o eliminar plantillas de forma sencilla. Esot permite que el desarrollo de plantillas se maneje de forma independiente al resto de la aplicacion y que se puedan añadir nuevas plantillas sin necesidad de modificar el codigo de la aplicacion.

7.1. Añadir Plantillas

Existiendo varias formas en las que se pudiera haber diseñado la forma con la que añadir nuevas plantillas a la aplicacion aprovechando caracteristicas de Github se ha optado por la solucion más sencilla y directa. Añadir una nueva plantilla es tan sencillo como añadir una nueva carpeta al repositorio de plantillas de GitHub. Esta carpeta debe contener una estructura de directorios especifica que contenga los archivos necesarios para la plantilla como son el archivo uvl, las plantillas Jinja2 y el archivo de Mapping Model, este ultimo archivo es opcional. Existen instucciones detalladas de como añadir una plantilla al respositorio de plantillas en CITA

7.1.1. Ampliar la Plantilla Dockerfile

La Plantilla Dockerfile recoje alguna de las caracteristicas más interesantes que ya cubren otras herramientas pero queda claro que todavia existe mucho margen de mejora en cuanto a la personalizacion de las opciones de configuracion. La plantilla no es a dia de hoy capaz de cubrir todas las opciones de configuracion que se pueden dar en un fichero Dockerfile, pero tiene el potencial de ser ampliada y mejorada en el futuro. Ampliar esta plantilla es sencillo,

se cuenta ya con un repositorio de desarrollo para la plantilla CITA que contiene la plantilla y los test necesarios para validarla. Una vez que se haya ampliado la plantilla, se puede añadir al repositorio de plantillas de GitHub y utilizarla para generar productos.

8

Despliegue de la aplicación

8.1. Despliegue local

Para desplegar el proyecto y todas sus capas se hace uso de la herramienta Docker CITA, que permite la creación de contenedores ligeros y portables que pueden ejecutarse en cualquier entorno. Para el despliegue local se hace uso de Docker Compose CITA, que permite definir y ejecutar aplicaciones Docker multi-contenedor generando todos los servicios necesarios para el despliegue de la aplicación. Esto supone una abstracción de la infraestructura necesaria para el despliegue de la aplicación, permitiendo la creación de un entorno de desarrollo y pruebas aislado y reproducible. En el proyecto se encuentra el fichero docker-compose.yml que contiene la configuración necesaria para el despliegue de la aplicación en un entorno local.

```
version: '3.8'

services:
  fronted:
 build:
   context: ./Frontend
   dockerfile: dockerfile
 container_name: frontend
 ports:
   - "4200:80"
```

```
 repository-manager:
build:
  context: ./Repository-Manager
  dockerfile: dockerfile
container_name: repository-manager
ports:
  - "5000:5000"

 uvengine-resolver:
build:
  context: ./UVEngine-Resolver
  dockerfile: dockerfile
container_name: uvengine-resolver
ports:
  - "5001:5001"
depends_on:
  - repository-manager

 nginx-reverse-proxy:
build:
  context: ./Nginx-Reverse-Proxy
  dockerfile: dockerfile
container_name: nginx-reverse-proxy
ports:
  - "80:80"
```

Como se observa el codigo se compone de 4 servicios

- Frontend: Servicio que se encarga de servir la aplicacion web
- Repository Manager: Servicio que se encarga de servir los archivos uvl de las plantillas y las versiones disponibles

- UVEngine Resolver: Servicio que se encarga de resolver la variabilidad a partir de recibir como parametros Un fichero Json que contiene el nombre de la plantilla, la version a utilizar y la configuracion
- Nginx Reverse Proxy: Servicio que se encarga de redirigir las peticiones de la aplicacion web al servidor de backend y fronte

La instruccion dockercompse up creara los contendores en la configuracion ya escrita. La web sera accesible en la direccion localhost en el puerto 80

8.2. Despliegue en la nube

Por definir

9 Resultados

Esta parte del documento se centra en los resultados obtenidos durante el desarrollo del proyecto, incluyendo los resultados de las pruebas, la viabilidad del proyecto, y la escalabilidad.

9.1. Resultados Obtenidos

Cumpliendo los objetivos propuestos al inicio del proyecto, se ha desarrollado una aplicación que permite la generación automática de archivos de configuración a partir de plantillas.

Un resumen de los resultados del proyecto es el siguiente:

- La Plataforma web de la aplicacion aporta una forma agil de crear productos software a partir de ellas.
- Las plantillas para generar ficheros como dockerfiles recogen una un interesante número de posibilidades entre las que se encuentran las opciones más usuadas una vez generamos una configuracion podemos simplemente descargar el fichero generado o copiarlo al portapapeles y quedaria listo para su uso.
- El empleo de la extension uvls para generar una configuracion puede ser muy eficiente a la hora de generar productos con opciones de configuracion relacionadas entre si Ademas tenemos la posiblidad de editar el producto final antes de utilizarlo.
- Las plantillas exitentes aportan unas funcionalidades interensates que pueden explotarse a la hora de genera productos como ficheros de Configuracion
- El repositorio es facilmente ampliable para abarcar nuevas plantillas y desarrollar nuevas funcionalidades

Resumen de los desarrollos resultantes del proyecto:

- Se ha desarrollado una aplicación web que permite la interacción con el usuario y la visualización de los archivos de configuración generados.
- Se adaptado el uso del proyecto uvengine que permite la generación automática de archivos de configuración a partir de plantillas adaptandola a las necesidades del proyecto.
- Se ha desarrollado una aplicación de Repository Manager que permite la comunicación con los repositorios de plantillas en GiHub CITA .
- Se ha desarrollado una aplicación de UVEngine Resolver que permite la resolución de la variabilidad a partir de una configuración.
- Se ha creado la plantilla para la generación de ficheros Dockerfile a partir de una configuración.
- Se ha desarrollado la plantilla auxiliares para ficheros de configuracion Nginx, dockerignore, etc.
- Se han desarrollado pruebas para validar la correcta construcción de los productos a partir de las plantillas dockerfile.
- Se ha aportado todo el código fuente y la documentación necesaria para la replicación del proyecto.
- Se ha generado una Memoria que documenta todo el proceso de desarrollo.
- Se ha aportado apendices para el uso más fundamental de la aplicacion.

9.2. Viabilidad del proyecto

Como hemos recogido arriba el proyecto es potencialmente viable pero presenta algunas carencias que pueden ser corregidas. Es posible que el uso mandatorio de la extension uvls para generar los ficheros de configuracion sea un punto en contra para la viabilidad del proyecto, pese a que la extension es un proyecto de codigo abierto y seria posible su integracion en la aplicacion. Las soluciones actualmente actualemente disponibles son más comodas al detectar

las features CITA del proyecto de forma autonoma sin necesidad de intervencion del usuario. Esto supone que esta solucion quede atras en cuanto a comodidad. Es conveniente que adaptar este sistema para que sea capaz de detectar algunas de las features de forma autonoma y dejar otras para la personalizacion del usuario al menos en cuanto nos referimos a la generacion del fichero dockerfile.

9.3. Escalabilidad

Teniendo en cuenta que la escalabilidad de cualquier aplicacion depende en la capacidad de sus servicios de generar nuevas instacias es en un principio viable escalar horizontalmente cada uno de los servicios. Sin embargo la escalabilidad de la aplicacion se ve limitada actualmente por el uso de API publica de github CITA para el servicio de Repository manager CITA que limita el numero de peticiones que se pueden hacer en un periodo de tiempo. [16] En el escenario en el que se quiera llevar la aplicacion a un entorno de produccion debe sortearse este problema bien por medio del uso de tokens de autenticacion. Existe la posbilidad de que el usario aporte un token personal de una cuenta de gitHub a la hora de desplegar el servicio o bien emplear un token propio desde Github.

En el escenario de superar dicho limite desde la aplicacion no se podra descargar modelos uvl ni se podra recargar el listado de plantillas y ni versiones de las mismas. No obstante se podra resolver la variabilidad de las plantillas que ya se han descargado y se podra generar productos a partir de ellas si se encuentran disponibles en los servicios uvengine-resolver CITA

[16] Actualmente el número de peticiones que puede realizarse a la api son 80 peticiones por minuto CITA esto es propenso a que cambien en un futuro.

10

Conclusiones y líneas Futuras

10.1. Conclusiones del Desarrollo del proyecto

El desarrollo del proyecto ha permitido alcanzar los objetivos planteados inicialmente. Se ha logrado implementar una solución eficiente y potencialmente escalable que cumple con los requisitos funcionales y no funcionales establecidos. Además, se ha adquirido un conocimiento profundo sobre las tecnologías utilizadas y se han identificado áreas de mejora para futuros desarrollos. El desarrollo de las plantillas ha sido conservador y no se ha considerado explorar más que casos basicos de uso. Esto era algo esperado si asumimos la gran casuistica que tenemos, sin embargo se ha demostrado que la aplicacion es capaz de generar productos validos a partir de configuraciones validas. Dado el nivel de desarollo de las herramientas (uvls) y tecnologias de las que hace uso este proyecto se ha demostrado que es posible llevar a cabo un proyecto de estas caracteristicas.

10.2. Conclusion Personal

A nivel personal, el desarrollo de este proyecto me ha servido para introduccirme de lleno en la tecnologia de contenedores, como esta tecnologia se aplica en servicios y que implicaciones existen a nivel de rendimiento y seguridad. Considero que el proyecto tiene de base una serie de fortalezas que otra serie de herramientas no consigue ofrecer actualmente. Es posible que un proyecto más ambicioso y con más recursos pueda explotar estas fortalezas y llevar el diseño de productos por este camino. Sin duda el proyecto queda lejos de tener aplicaciones funcionales reales pero muestra un camino que puede ser explotado si se considera.

10.3. Aplicaciones Prácticas

Ademas de contar con una forma intersante de generar dockerfiles pueden existir dinamicas para aprovechar configuraciones validas para generar otros ficheros necesarios, como por ejemplo podemos ficheros de configuracion para el servidor de Nginx CITA directamente a partir de la configuracion para generar un dockerfile. Esto es gracias a los ficheros Mapping Models y a un diseño inteligente de las plantillas

10.4. Desarrollo de nuevas plantillas

El proyecto esta preparado para recibir nuevas plantillas no solo para la generacion de ficheros dockerfile y algunos de los archivos de configuracion sino que se puede extender a otros tipos de ficheros, como por ejemplo ficheros de configuracion de Kubernetes CITA, ficheros de configuracion de Jenkins CITA, etc. Una vez añadidas plantillas al repositorio estan estaran disponibles de la misma forma en la que estan disponibles las plantillas existentes solo sera necesario recargar el listado de plantillas que se encuentra en el la barra de navegacion de la aplicacion web.

10.5. Continuación en el desarrollo de plantillas ya existentes

Las plantillas exitentes tienen un gran margen de mejora, En la situacion actual no soportan la gran mayoria de las opciones de configuracion que se pueden dar en un fichero Dockerfile. Además las opciones que se manejar son propensas a quedar obsoletas y se necesita de una continua revision de las plantillas en busca de problemas de diseño o seguridad.

10.6. Mejorar la experiencia del usuario

La generacion de la configuraciones estan ligadas al desarrollo de la extension uvls, la extension permite al usuario generar una configuracion valida deplegando el arbol de caracteristicas CITA pero para ello el usuario necesita contar con VSCode y la extension instalada. para ser capaz de generar configuraciones validas. La propuesta para este escenario seria la integracion de dicha funcionalidad dentro de la plantaforma web, la extension uvls es un proyecto de codigo abierto y su integracion en la aplicacion web podria ser un proyecto en si

mismo.

En el desarrollo del trabajo se ha explorado esta posibiliddad ejecutando dicha extension en la version web del editor VS code pero el resultado en cuanto a rendimiento hace que esta opcion no sea viable.

La propia aplicacion web puede ser mejorada en cuanto a la experiencia del usuario añadiendo más informacion al usuario para colocar más contexto a la hora de seleccionar una plantilla y las posiblidades de combianar configuraciones

Referencias

[1] Amazon Web Services. *The Difference Between Docker Images and Containers*. Accessed: 2024-10-21. 2024. URL: https://aws.amazon.com/es/compare/the-difference-between-docker-images-and-containers/#:~:text=Una%20imagen%20de%20Docker%2C%20o,el%20contenedor%20necesita%20para%20ejecutarse..

[2] Angular. *Angular - The modern web developer's platform*. Accedido: 22 de octubre de 2024. 2024. URL: https://angular.dev/.

[3] Angular. *Angular Material*. Accedido: 22 de octubre de 2024. 2024. URL: https://material.angular.io/.

[4] Caradhras. *UVLS - Universal Variability Language Support*. Accedido: 22 de octubre de 2024. 2024. URL: https://marketplace.visualstudio.com/items?itemName=caradhras.uvls-code.

[5] Visual Studio Code. *Developing inside a Container*. Accedido: 21 de octubre de 2024. 2024. URL: https://code.visualstudio.com/docs/containers/overview.

[6] UVL Community. *Universal Variability Language (UVL)*. Accedido: 21 de octubre de 2024. 2024. URL: https://universal-variability-language.github.io/.

[7] Docker. *Docker Desktop*. Accedido: 21 de octubre de 2024. 2024. URL: https://www.docker.com/products/docker-desktop/.

[8] Docker. *Dockerfile Reference*. Accedido: 22 de octubre de 2024. 2024. URL: https://docs.docker.com/reference/dockerfile/.

[9] Inc. Docker. *Dockerfile Concepts*. Accessed: 2024-10-21. 2023. URL: https://docs.docker.com/build/concepts/dockerfile/.

[10] Inc. Docker. *Intro Guide to Dockerfile Best Practices*. Accessed: 2024-10-21. 2023. URL: https://www.docker.com/blog/intro-guide-to-dockerfile-best-practices/.

[11] Docker, Inc. *Docker: Accelerate How You Build, Share, and Run Modern Applications*. Accessed: 2024-10-21. 2024. URL: https://www.docker.com/.

[12] Python Software Foundation. *Python*. Accedido: 22 de octubre de 2024. 2024. URL: https://www.python.org/.

[13] Inc. GitHub. *GitHub*. Accedido: 22 de octubre de 2024. 2024. URL: https://github.com/.

[14] Inc. GitHub. *GitHub REST API Documentation*. Accedido: 22 de octubre de 2024. 2024. URL: https://docs.github.com/en/rest?apiVersion=2022-11-28.

[15] Google Cloud. *What are Containers?* Accessed: 2024-10-21. 2024. URL: https://cloud.google.com/learn/what-are-containers?hl=es.

[16] Gunicorn. *Gunicorn*. Accedido: 22 de octubre de 2024. 2024. URL: https://gunicorn.org/.

[17] IBM. *Orquestación de contenedores*. Accedido: 21 de octubre de 2024. 2024. URL: https://www.ibm.com/es-es/topics/container-orchestration.

[18] Panamerican Latam. *¿Qué es un prompt en IA y para qué sirve?* Accedido: 21 de octubre de 2024. 2024. URL: https://panamericanlatam.com/que-es-un-prompt-en-ia-y-para-que-sirve/.

[19] MagickPen. *Plantilla para generar Dockerfiles*. Accedido: 21 de octubre de 2024. 2024. URL: https://magickpen.com/templates/128/.

[20] O'Reilly Media. *Docker Certified Associate*. Accedido: 22 de octubre de 2024. 2024. URL: https://www.oreilly.com/library/view/docker-certified-associate/9781839211898/c5ecd7bc-b7ed-4303-89a8-e487c6a220ed.xhtml#uuid-1a5da664-fb76-4e56-bdb0-83255dde9e78.

[21] Microsoft. *Monaco Editor*. Accedido: 22 de octubre de 2024. 2024. URL: https://microsoft.github.io/monaco-editor/.

[22] Microsoft. *Visual Studio Code*. Accedido: 22 de octubre de 2024. 2024. URL: https://code.visualstudio.com/.

[23] Inc. Nginx. *Nginx*. Accedido: 22 de octubre de 2024. 2024. URL: https://nginx.org/en/.

[24] Node.js. *Node.js*. Accedido: 22 de octubre de 2024. 2024. URL: https://nodejs.org/en/.

[25] Inc. npm. *ngx-json-viewer*. Accedido: 22 de octubre de 2024. 2024. URL: https://www.npmjs.com/package/ngx-json-viewer.

[26] Inc. npm. *npm*. Accedido: 22 de octubre de 2024. 2024. URL: https://www.npmjs.com/.

[27] OpenAI. *ChatGPT*. Accedido: 21 de octubre de 2024. 2024. URL: https://chatgpt.com/.

[28] Podman Project. *Podman: A Tool for Managing Containers and Pods*. Accessed: 2024-10-21. 2024. URL: https://podman.io/.

[29] Pallets Projects. *Flask Documentation (3.0.x)*. Accedido: 22 de octubre de 2024. 2024. URL: https://flask.palletsprojects.com/en/3.0.x/.

[30] Pallets Projects. *Jinja Documentation (3.1.x)*. Accedido: 22 de octubre de 2024. 2024. URL: https://jinja.palletsprojects.com/en/3.1.x/.

[31] PyPI. *Python Package Index*. Accedido: 22 de octubre de 2024. 2024. URL: https://pypi.org/.

[32] Git SCM. *Git*. Accedido: 22 de octubre de 2024. 2024. URL: https://git-scm.com/.

[33] *UVL Playground*. Accedido: 22 de octubre de 2024. 2024. URL: https://uvl.uni-ulm.de/.

[34] Wikipedia. *Feature Model*. Accedido: 21 de octubre de 2024. 2024. URL: https://en.wikipedia.org/wiki/Feature_model.

Apéndice A
Manual de Instalación

A.1. Requisitos Previos

A.2. Descargar el codigo fuente

A.3. Configuracion del archivo Dockercomose

A.3.1. Nginx reverse proxy

A.3.2. frontend

A.3.3. Repository Manager

A.3.4. UV Engine resolver

A.4. Construccion de los contenedores

A.4.1. Github API

A.5. Terminar el servicio

Apéndice B
Manual de Uso

B.1. Inicio de la aplicacion

B.2. Barra de Navegación

B.2.1. About Page

B.2.2. Help Page

B.3. Seleccionar plantilla

B.3.1. Seleccionar version de la platilla

B.3.2. Cargar configuracion

B.3.3. Descargar Fichero Generado

B.3.4. Copiar Fichero Generado

Apéndice C
Manual de Desarrollo

C.1. Manual de Desarrollo de Plantillas

C.1.1. Como añadir nuevas plantillas al repositorio

C.1.2. Como modificar las plantillas del repositorio

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA