ANEXO III – DOCUMENTACIÓN TÉCNICA

ARQUITECTURA DE SERVICIOS RESTful
PARA UN JUEGO DE TABLERO ONLINE

Trabajo de Fin de Máster

MÁSTER EN INGENIERÍA INFORMÁTICA

VNiVERSiDAD
D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

Febrero de 2023

AUTOR

Javier Vidal Ruano

TUTOR

Rodrigo Santamaría Vicente

Índice de contenido

1 – Introducción

El presente documento tiene como objetivo servir como una guía para los desarrolladores en cuanto a la lectura y modificación del código del sistema.

La estructura del código del sistema es la que se muestra a continuación:

- “documentation”: Directorio que contiene toda la documentación del proyecto.
- “specification”: Directorio que contiene la especificación técnica en formato OpenAPI 3.0 de los servicios del sistema, además de las colecciones de peticiones de Postman para los mismos.
- “src”: Directorio que contiene el código de los servicios del sistema y todo lo necesario para realizar el despliegue.

2 - Documentación

Bajo el directorio “documentation” hay dos subdirectorios:

- “reports”: Directorio que contiene los informes relativos al proyecto en formato PDF.
- “static-html-documentation”: Directorio que contiene la documentación de los servicios del sistema en formato de página web. Para visualizarlas basta con abrir el archivo “index.html” desde el navegador.

3 – Especificación de los servicios

Bajo el directorio “specification” hay los siguientes archivos:

- “OpenAPI3.0-specification-resources.yaml”: Archivo que contiene la especificación del servicio de recursos en formato OpenAPI 3.0.
- “OpenAPI3.0-specification-control.yaml”: Archivo que contiene la especificación del servicio de control en formato OpenAPI 3.0.
- “postman-collection-resources.json”: Archivo que contiene la colección de llamadas de Postman para todos los endpoints del servicio de recursos. Se encuentra lista para importarla y comenzar a usarla.
- “postman-collection-control.json”: Archivo que contiene la colección de llamadas de Postman para todos los endpoints del servicio de control. Se encuentra lista para importarla y comenzar a usarla.

4 – Código del proyecto

Bajo el directorio “src” se encuentra todo el código del proyecto. Contiene los siguientes archivos y directorios:

- “resources”: Directorio que contiene todo el código relativo al servicio de recursos.
- “control”: Directorio que contiene todo el código relativo al servicio de control.
- “nginx”: Directorio que contiene el archivo de configuración del servidor Nginx usado para el despliegue del sistema, además de un script en Bash para generar un certificado autofirmado, necesario para permitir una comunicación por HTTPS.
- “docker-compose.yaml”: Archivo que contiene la configuración de Docker Compose que permite desplegar y manejar el sistema con unos sencillos comandos. Entre otros:
    - “docker-compose up -d”: Levanta los contenedores del sistema, exponiendo únicamente el puerto 443 en el equipo anfitrión.
    - “docker-compose down”: Para los contenedores del sistema y los elimina.
    - “docker-compose restart”: Reinicia los contenedores del sistema.

4.1 – Directorio “resources”

Este directorio contiene los siguientes archivos:

- “requirements.txt”: Archivo que contiene las dependencias de Python.
- “openapi.json”: Archivo que contiene la especificación del servicio en bajo el estándar OpenAPI 3.0, en formato JSON.
- “launch.sh”: Pequeño script que permite lanzar el servicio de forma aislada, escuchando peticiones en el puerto 8000.
- “Dockerfile”: Archivo que permite generar la imagen de Docker del servicio.
- “multimeta.py”: Archivo Python que contiene una clase que permite realizar sobrecarga de métodos. Se usa en los archivos “board.py” y “cards.py”.
- “validators.py”: Archivo Python que contiene los modelos de Pydantic, usados para validar la entrada del usuario en los endpoints del servicio.

- “main.py”: Archivo Python que supone el punto de entrada del servicio. Se encarga de crear el servicio con FastAPI y de definir los endpoints. Crea instancias de la clase Game.
- “game.py”: Archivo Python donde se define la clase Game, usada para representar las partidas del usuario. Crea instancias de las clases Board y Cards.
- “board.py”: Archivo Python donde se define la clase Board. Utiliza el archivo “board.json” para inicializar los valores de cada instancia.
- “board.json”: Contiene los valores iniciales del tablero de juego.
- “cards.py”: Archivo Python donde se define la clase Cards. Utiliza el archivo “cards.json” para inicializar los valores de cada instancia.
- “cards.json”: Contiene los valores iniciales de las cartas.

4.2 – Directorio “control”

Este directorio contiene los siguientes archivos:

- “requirements.txt”: Archivo que contiene las dependencias de Python.
- “openapi.json”: Archivo que contiene la especificación del servicio en bajo el estándar OpenAPI 3.0, en formato JSON.
- “launch.sh”: Pequeño script que permite lanzar el servicio de forma aislada, escuchando peticiones en el puerto 8001.
- “Dockerfile”: Archivo que permite generar la imagen de Docker del servicio.
- “multimeta.py”: Archivo Python que contiene una clase que permite realizar sobrecarga de métodos. Se usa en el archivo “game.py”.
- “validators.py”: Archivo Python que contiene los modelos de Pydantic, usados para validar la entrada del usuario en los endpoints del servicio.
- “main.py”: Archivo Python que supone el punto de entrada del servicio. Se encarga de crear el servicio con FastAPI y de definir los endpoints. Crea instancias de la clase Game.
- “game.py”: Archivo Python donde se define la clase Game, usada para representar las partidas del usuario y que contiene todas las operaciones que este puede realizar.
- “config.py”: Archivo Python que se encarga de cargar los valores de las variables de entorno definidas en el archivo “.env”.
- “.env.example”: Archivo que debe usarse como plantilla para crear el archivo “.env”, especificando las variables de entorno necesarias para el correcto funcionamiento del servicio de control. Sirve para lanzar el servicio sin tener que ejecutar el Docker Compose.