



SOLICITUD ALUMNO/A EN PRÁCTICAS

Centro	Instituto del Corcho, la Madera y el Carbón Vegetal
Investigador/a (tutor/a)	Adrián Javier Montero Calvo
Departamento/ Área	Área de Sistemas Forestales Mediterráneos
Nº plazas a solicitar	1
Período	400 horas entre enero y junio
Formación/ Estudios preferentes	Ciclo Formativo de Grado Superior (CFGS) en Desarrollo de Aplicaciones Multiplataforma
Tema de trabajo	Endpoint para Base de datos
Breve descripción de tareas a desarrollar	Se dispone de una réplica a tiempo real de una base de datos donde se guardan de forma automática datos de sensores asociados a un cultivo forestal. Durante el periodo formativo el alumno en práctica desarrollará una aplicación containerizada que permita la extracción de datos de la misma mediante la creación de una API REST bien definida. Las tareas a desarrollar serán: <u>Configuración del Entorno y Preparación del Proyecto</u> * Análisis de la base de datos: Explorar la réplica de la base de datos para entender el esquema de las tablas, los tipos de datos de los sensores (temperatura, humedad, luz, etc.) y las relaciones entre ellos. * Selección de tecnologías y Gestión de dependencias. * Configuración de la conexión: Establecer y probar la conexión segura a la réplica de la base de datos, utilizando variables de entorno para gestionar las credenciales de forma segura. <u>Desarrollo de la API REST</u> * Diseño de la API: Definir los endpoints (rutas) de la API REST de forma clara y lógica, siguiendo las buenas prácticas (ej. uso de sustantivos, verbos HTTP correctos). * <code>`GET /sensores`</code>: Obtener una lista de todos los sensores. * <code>`GET /sensores/{id}`</code>: Obtener los datos de un sensor específico por su ID. * <code>`GET /sensores/{id}/datos`</code>: Obtener los datos históricos de un sensor. * Implementación de la lógica de negocio: Escribir las funciones que interactúan con la base de datos para ejecutar consultas y recuperar los datos. * Manejo de consultas: Implementar la lógica para procesar parámetros de consulta en la URL, como filtrado (ej. por fecha <code>`?start_date=...&end_date=...`</code>) y paginación (<code>`?page=...&limit=...`</code>). * Serialización de datos: Convertir los datos recuperados de la base de datos a un formato estándar como JSON antes de enviarlos como respuesta. * Gestión de errores: Implementar un manejo robusto de errores para devolver códigos de estado HTTP junto con mensajes de error descriptivos. <u>Containerización y Despliegue</u> * Creación del Dockerfile: Escribir un <code>`Dockerfile`</code> para definir la imagen del contenedor, especificando la imagen base, copiando los archivos del proyecto, instalando las dependencias y definiendo el comando para ejecutar la aplicación. * Construcción de la imagen: Ejecutar el comando <code>`docker build`</code> para crear la imagen del contenedor de la aplicación. * Ejecución del contenedor: Probar la aplicación en un contenedor aislado con el



	comando `docker run`, mapeando los puertos necesarios. * Orquestación Crear un archivo `docker-compose.yml`. <u>Pruebas y Documentación</u> * Pruebas unitarias: Escribir pruebas automatizadas para validar que cada función y endpoint de la API funciona como se espera. * Pruebas de integración. * Documentación de la API: Utilizar una herramienta (ej. Swagger/OpenAPI) para generar una documentación interactiva. * Control de versiones: Subir el proyecto a un repositorio Git (Codeberg).
--	--