Parcial 2. Servicios Web

El *crowdsourcing* consiste en externalizar tareas a través de una convocatoria abierta, dejándolas a cargo de una comunidad dispuesta a colaborar en ellas. De este modo, es posible recopilar servicios, ideas o contenido a través de las contribuciones de un gran número de personas.

En este ejercicio desarrollarás una API REST para *Gente!*, una aplicación web de *crowdsourcing*, en la que se pueden gestionar tareas que serán particionadas entre un grupo de colaboradores, que llevan a cabo las distintas partes o segmentos de la misma.

1. Diseño de la base de datos (1,5 puntos)

En el modelo conceptual de *Gente!* distinguiremos dos tipos de elementos:

- ***Tareas.*** Las tareas en las que se puede colaborar, descritas por las siguientes características:
  - ***responsable:*** dirección de *email* del usuario responsable de la tarea (el que crea la tarea).
  - ***descripción:*** título o descripción breve de la tarea (hasta 50 caracteres).
  - ***habilidades:*** una serie de habilidades (términos) adecuadas para participar en la tarea.
  - ***segmentos:*** duración estimada de la tarea (en segmentos de 1 hora de trabajo).
- ***Colaboradores.*** Representa a los usuarios de la aplicación, con las siguientes características:
  - ***email:*** dirección de *email* del colaborador.
  - ***nombre:*** nombre del usuario.
  - ***habilidades:*** una serie de habilidades (términos) que posee el colaborador.

Como se ha indicado, varios colaboradores participan en el desarrollo de una tarea, cada uno de ellos llevando a cabo uno de los segmentos de la misma. Un colaborador puede participar en varias tareas. Una misma persona puede figurar como responsable de varias tareas.

Para el almacenamiento de información, utilizarás una base de datos no relacional. Por tanto, los elementos del modelo conceptual pueden dar lugar a un número mayor o menor de entidades de base de datos, con más o menos atributos, dependiendo de la base de datos elegida y el diseño creado para representarlos.

2. Servicios REST básicos (2 puntos)

Una vez diseñada la estructura de la base de datos, desarrollarás un proyecto que despliegue localmente una API REST cuyas operaciones devolverán sus resultados en JSON. No es necesario que desarrolles varios microservicios.

Se valorará especialmente que los *endpoints* del servidor y sus posibles parámetros estén definidos de acuerdo a las normas y recomendaciones de estilo propias de los servicios REST.

El API REST ofrecerá las siguientes operaciones básicas:

- CRUD de tareas: GET ALL, GET, POST, PUT, DELETE.
- CRUD de colaboradores: GET ALL, GET, POST, DELETE. Observa que no hay operación PUT.
- CRUD de habilidades de un colaborador: GET ALL (devuelve todas sus habilidades), POST (añade una habilidad al colaborador), DELETE (elimina una habilidad). Observa que no hay operación PUT, ni GET individual.

3. Servicios REST adicionales (3 puntos)

Ampliarás la funcionalidad de la API REST con las siguientes operaciones:

- Tareas que requieren una determinada habilidad, devolviendo una lista de tareas.
- Tareas asignadas a un determinado colaborador, devolviendo una lista de tareas.
- Asignar un colaborador a una tarea, para lo cual se comprobará que el colaborador posea al menos una de las habilidades requeridas por la tarea.
- Buscar candidatos a colaborar en una tarea, devolviendo una lista de emails de colaboradores que posean al menos una de las habilidades requeridas por la tarea.
- Tareas completamente asignadas, que devolverá las tareas que tengan asignados tantos colaboradores como segmentos de los que consta la tarea.
- Buscar los colaboradores de un determinado usuario, devolviendo una lista de emails de colaboradores que participen en alguna de las tareas de las que el usuario es responsable.

4. Descripción y pruebas del servicio (1,5 puntos)

Las operaciones de la API REST ofrecerán una interfaz coherente que permita la gestión de tareas mediante *crowdsourcing*. Todas las operaciones del servidor se podrán probar de forma independiente. Las pruebas pueden definirse a partir de una especificación *OpenAPI* o un conjunto de pruebas *Postman*, que acompañarán a la entrega del examen. Tanto en un caso como en otro, serán lo suficientemente descriptivas (incluyendo los comentarios y ejemplos que consideres necesarios) como para permitir la prueba del servidor de forma sencilla.

5. Despliegue del servicio (2 puntos)

El despliegue de la API REST (y de la base de datos, si es local) se realizará con *Docker*. En la entrega del examen incluye los *scripts Docker* para realizar dicho despliegue, junto con cualquier archivo de configuración (ej., `.env` para definir variables de entorno) y de dependencias (ej., `requirements.txt` para instalar paquetes en Python) necesarios para poder realizar dicho despliegue mediante `docker compose up` desde la carpeta raíz del proyecto. Indica en la memoria de la entrega si fuese necesario realizar otras acciones distintas para el despliegue.

Entrega

Entregarás este ejercicio a través del campus virtual, mediante un archivo comprimido[i] que contenga:

- Una **memoria breve** en la que describas:
  - el diseño de la base de datos (apartado 1) describiendo su estructura (entidades y atributos) y su URI o credenciales de acceso. Si la base de datos es local, adjunta en la entrega una descarga o *backup* de la misma.
  - las tecnologías utilizadas (lenguaje de programación, *frameworks*, etc.), y las instrucciones de instalación y despliegue, así como el puerto de despliegue del servidor.
  - las limitaciones de la solución propuesta y los problemas encontrados durante su desarrollo (si procede).
- Las **fuentes** completas del servidor REST con las operaciones desarrolladas (apartados 2 y 3), así como los archivos de entorno y dependencias (si procede) y los *scripts Docker* necesarios para su despliegue (apartado 5).
- La **especificación OpenAPI** o un conjunto de **pruebas Postman** del servicio (apartado 4).

[i] Para subir el código de la aplicación al campus, elimina primero el entorno virtual que hayas utilizado (si tu solución está en Python) o la carpeta `node_modules` (si tu solución está en Node.js) o cualquier otro archivo que se genere automáticamente al desplegar el sistema y que haga que el archivo comprimido tenga un tamaño excesivo (y por tanto, mucho tiempo de compresión y de subida al campus).