RAZONAMIENTO LOGICO:

EJERCICIO # 1: COMPLETAR LA SECUENCIA.

1. RQP, ONM, LKJ, _____, FED a. IHG b. CAB c. JKL d. GHI	**2. KBJ, LCK, MDL, NEM, _____** a. OEP b. OFN c. MEN d. PFQ
3. 104, 109, 115, 122, 130, _____ a. 119 b. 125 c. 139 d. 134	**4. 15, 31, 63, 127, 255, _____** a. 110 b. 170 c. 511 d. 181

EJERCICIO 2: ¿DESEA BEBER ALGO?

Dado un grupo de 5 personas: A, B, C, D, E. Se tiene:

i) Una de ellas esa artista, otra es médico, otra es periodista, otra es deportista y otra es juez.
ii) Tres de ellas A, C y el juez, prefieren beber té, B y el periodista prefieren beber café.
iii) El deportista, D y A, son amigos entre sí, pero dos de ellos prefieren el café.
iv) El artista es hermano de C.

Sabiendo esto responde:

1. ¿Quién es el Artista?
   R/ A es el artista
2. ¿Quién es el Deportista?
   R/ B es el deportista
3. ¿Quién es el Medico?
   R/ C es el medico
4. ¿Cuál de los siguientes grupos incluye a una persona que prefiere el té pero que no es el juez?
   a. A-C-E
   b. D-E
   c. B-C-E
   d. B-D
   e. Ninguno de los anteriores

R/La respuesta es la c ya que este es el único que incluye una persona que prefiere beber té y que no sea el juez, ya que la E es el juez, la B es el deportista y prefiere café y la C es el médico y también prefiere el té.

PRUEBA TECNICA DESARROLLADORES

Condiciones de la prueba

- Tiempo de implementación: 6 Horas.
- Se acepta presentación en pseudocódigo.
- Sin restricción de lenguaje (**Preferiblemente C#**).
- Solo se permite el uso de framework frontend (como bootstrap, handlebars, kendoUI etc.).

Requerimiento

Una serie de excursionistas desea crear un software que permita determinar los elementos que puedan usarse para escalar un risco basado en las propiedades calóricas y peso de cada elemento, para escalar el risco se especificara la cantidad mínima de calorías además del peso máximo que se puede llevar. El software deberá de indicar al usuario el conjunto de elementos óptimos basado en que cumplan el mínimo de calorías requeridas y el llevar el menor peso posible.

Un ejemplo es:

Mínimo de calorías: 15
Peso máximo: 10

Elementos:

E1 Peso 5 Calorías: 3
E2 Peso 3 Calorías: 5
E3 Peso 5 Calorías: 2
E4 Peso 1 Calorías: 8
E5 Peso 2 Calorías: 3

En esto caso elementos viables son: **E1 E2 E4**, ya que su peso total seria 9 y brindan 16 calorías.

Requerimientos opcionales de solución:

1. La aplicación debe ser interoperable.
2. La aplicación debe ser de fácil mantenimiento.
3. El uso de control versión para la solución por ejemplo GitHub.
4. Persistencia de la información
5. Documente la escalabilidad de la solución.

Solución en código
https://github.com/eisazav/Mochilero

Escalabilidad de la solución

La solución está diseñada para escalar de manera horizontal y mantener un rendimiento óptimo aún bajo alta carga, gracias a las siguientes características:

1. Arquitectura Hexagonal (Ports & Adapters): Al utilizar una arquitectura hexagonal, separamos la lógica de negocio de las dependencias externas (como bases de datos, servicios de terceros o interfaces de usuario). Esto permite que cada componente evolucione de forma independiente y se pueda escalar o sustituir sin afectar el núcleo de

la aplicación. Además, favorece la implementación de pruebas unitarias y de integración, garantizando que la solución se mantenga robusta al crecer.

2. Uso de APIs RESTful: La aplicación se expone mediante APIs, lo que permite una comunicación desacoplada entre el frontend y el backend. Esto facilita la escalabilidad horizontal, ya que los endpoints pueden distribuirse en múltiples instancias y ser replicados en distintos servidores. Con un balanceador de carga, las peticiones se reparten eficientemente entre los servidores disponibles, evitando cuellos de botella y degradación del rendimiento en momentos de alta demanda.

3. Balanceo de Cargas y Escalabilidad Horizontal: La aplicación puede soportar balanceadores de carga, por ende, las solicitudes se pueden distribuir entre varias instancias, sino que también se beneficia de la posibilidad de escalar de forma horizontal. Esto significa que se pueden agregar más nodos al sistema, tanto en el backend (APIs) como en el frontend, lo que incrementa la disponibilidad y reduce las latencias. Esta capacidad es esencial para mantener una experiencia de usuario óptima ante picos de tráfico, ya que asegura que la carga se distribuya de forma eficiente y que ningún nodo se vea sobrecargado.