EXAMEN TECNICO GABRIELA CARRAZANA VIÑES

Solución en java script:

Se adjunta archivo: examentecnico.js

Para realizar este apartado he creado distintas funciones para seguir la lógica de negocio planteada en el apartado lo que resumiré a continuación

A partir de que recibo los dos archivos json, leo y parseo los datos de estos en 2 variables para luego procesarlos, de todos los prerepartos filtro los que solo cumplan con las condiciones de pertenecer a los grupos:

• grupoLocalizacionDesc = “CICLO 2 GRUPO A2” y “CICLO 1 GRUPO B” y “CICLO 1

GRUPO A2”

• esEcommerce = 1

A partir de esta lista más reducida de repartos proceso cada uno de ellos con el uso de la función **NewPetitionToStock** esta función contiene la lógica para satisfacer la propuesta. Luego, utiliza la función **ProcessRequest** para intentar satisfacer la propuesta utilizando el stock disponible. Finalmente, genera una tabla de resultados con la información relevante de reparto.

Otras Funciones: He creado funciones para manipular datos del stock ya que las cantidades de almacenamiento varían al satisfacer cada propuesta y he contemplado los casos en los que para una propuesta con key especificada no aparezcan coincidencias en el stock o aparezcan coincidencias, pero la cantidad total de stock no satisface la propuesta.

visualización de resultados: Como parte del ejercicio se pedía una tabla con las columnas:

Key (artículo a repartir)

2. idTienda

3. propuesta (unidades a repartir)

4. tipoStockDesc (zona del almacén de la que sale)

5. EstadoStock (nuevo campo que tendrá valores 1 o 5 según el estado del stock

que ha cogido [stockEm05 o stockEM01])

6. posicioncompleta (id de la posición en el almacén)

Además de estas he agregado 2 columnas:

TOTAL: contiene información acerca del total de unidades necesarias en el pedido que en caso de que con un solo stock no se satisfaga el mismo, será este campo distinto a propuesta.

COMENTARIOS: Para en el contemplar y brindar información para los casos en los que el pedido no se pueda completar por falta **STOCK INSUFICIENTE**: el stock existente no cubre completamente el pedido o ARTÍCULO **AGOTADO**: caso en el que en el stock no hay coincidencias para la key en cuestión, **COMPLETADO**: cuando el reparto es posible con el stock existente.

Solución LowCode:

Si tuviera que implementar esta propuesta como solución lowcode se me ocurren dos formas:

- Utilizando una API que admita endpoints para cargar y procesar los json de propuestas y stock.
- Otra solución, pero la menos escalable en caso de cambios es tener estos archivos. json en la carpeta de resourse para desde aquí manipularlos y procesarlos, considero esta función poco factible debido a que en este caso es una solución no muy eficiente para cambios en estos archivos, pero es otra manera de solucionar la problemática a modo de ejercicio, en un entorno real de trabajo es mucho más eficiente el consumo de API.

A partir de la obtención de los datos por vía API lo que realizaría sería las funciones necesarias equivalentes a las ya realizadas en code con JS para la manipulación de los datos, o bien el consumo de una API para procesar los datos.

Para generar la tabla, lo podría generar como una entidad de Outsystems y crear una interfaz de usuario para mostrarla por pantalla e incluso con funcionalidades de filtrado por Key, por zona de stock, estado de stock. También podría consumir una API que escriba los resultados procesados en la aplicación con code.

Con Outsystems realizaría el manejo de errores y validación de datos con el manejador de excepciones tanto al leer como al manipular y escribir datos, o bien pudiera también tener una API de manejo de errores y validación de datos. Debo tener en cuenta:

- ✓ Proporcionar códigos de estados HTTP apropiados y mensajes de error bien claros

Debo tener en cuenta Aspectos importantes de seguridad a la hora de consumir APIs como son:

- ✓ Usar mecanismo de autentificación y autorización para evitar accesos no autorizados a la API.
- ✓ Usar mecanismos de cifrado HTTPS para establecer la comunicación entre cliente y servidor de manera confiable.

- ✓ Desplegar la API en entorno de producción seguro.
- ✓ Monitorear el comportamiento de la API.

Como parte del proceso de implementación de este ejercicio debo realizar pruebas para verificar el correcto funcionamiento de este y una vez esté satisfecho con los resultados obtenidos, desplegar la aplicación en un entorno de producción fiable.

Si el json real del prerepato ocupase 20Gb, explica si el problema de forma distinta y por qué.

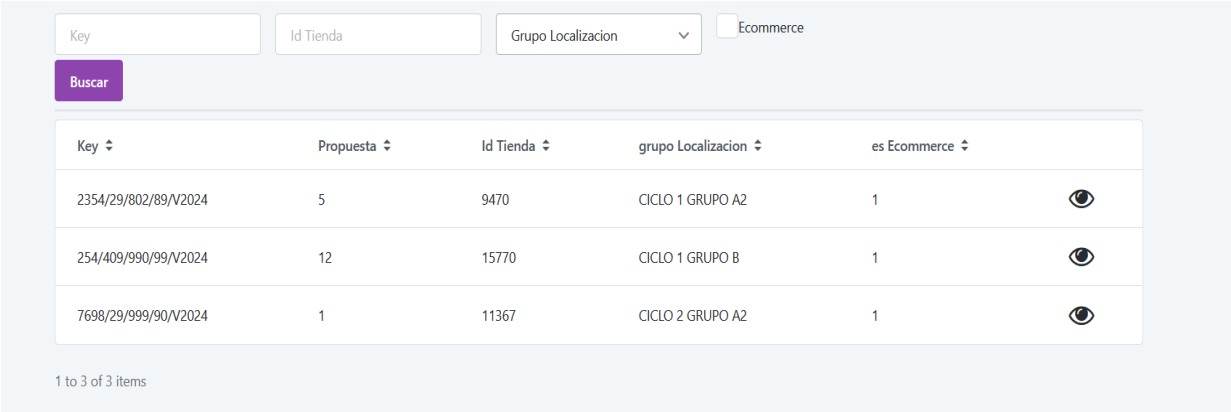
Estaríamos hablando de un gran volumen de datos por lo que considero que es necesario implementar un procesamiento de los mismos fraccionadamente para así no tener que manejar gran volumen de datos lo cual sería poco eficiente, también tendría en cuenta el uso de recursos de la nube ya que se estaría tratando de un gran volumen de información, la solución debería de propiciar escalabilidad y rendimiento capaz de afrontar crecimiento en el volumen de datos a manipular y procesar, el manejo de errores sería un aspecto muy relevante junto con la seguridad de los mismos por lo que al tratarse de un gran volumen de datos implementaría medidas de seguridad robustas. El proceso requerirá en este caso experiencia en el empleo de herramientas de bigData.

Si tuvieras que de forma visual presentar en una pantalla desde que partes de un almacén se rellena un pedido, que propuesta de visualización plantearías teniendo en cuenta que se quiere implementar con una herramienta low code

Propondría la siguiente opción de visualización que cuenta de una pantalla principal con filtros disponibles y la tabla de los repartos en la cual puedes filtrar según distintos criterios:

- ✓ Si es una ecommerce
- ✓ El TiendaId
- ✓ Key
- ✓ Grupo de localización desc

En función de estos filtros se modificaría el listado. Este sería el diseño propuesto para la pantalla de listado (Adjunto Imagen a solución):



Si se realiza click en el icono del ojo de una de las filas, pasamos a una pantalla de detalle del articulo para repartir seleccionado que contiene una tabla con la información de las distintas partes del stock donde hay que buscar el articulo a repartir, en este caso la tabla solo cuenta de una fila ya que el stock con su stockEM1 es capaz de cubrir la totalidad del pedido, pero la idea de este diseño es que aparezcan por pedido tantas filas como zonas del stock sean necesarias consultar para cubrir la propuesta: