Next important stepts

1. Crear el message log que es transaccional con el event store. Esto asegura que el rebuilding sea correcto SIEMPRE.
   1. Una forma de hacerlo es que el event store tome el mensaje **como parámetro** (una interfaz para que no tome un object plano) y luego dentro se lo castee para ver si es un evento o un comando y se loguee en el message log).
2. LIMPIAR EL COMMAND Y EVENT BUS, esto sirve para procesar los eventos con fecha pendientes. De esta forma se preserva el flujo de trabajos programados al momento de la reconstrucción. Que limpie el command bus y el event bus y lo ponga todo en 0 los generadores de id en la base de datos.
3. Hacer formatos empaquetados para colocar configuraciones con base de datos separados de Log, Engine. ReadModel. La web que hace los queryies debe tener pooling, mientras que el worker no.

Next steps

1. Capturar la exception que salta cuando colocamos un comando repetido en el bus. Hasta el momento aparece un DbUpdate Exception, pero lo podemos mejorar si hacemos que al saltar una excpcion se ejecute una consulta en el message log preguntando si existe un command con el mismo id. Si existe el mismo id entonces lanzar una exception customizada de duplicate message. Esto evitaría hacer que un DbUpdate exception que se deba a otro motivo no se oscurezca.

Conventions

Puntos de entrada para implementar un dominio con el Journey Framework:

1. Insertar el domain container en esta sección:
2. 

Arquitectura de un dominio. Assemblies = Layers

1. Domain: Aquí se encuentran los handlers y los aggregates
2. Domain.Commands: Aquí se encuentran los commands. Es el único lugar donde debería conectarse con la capa de aplicación.

Reglas:

1. Todo comando debe generar un evento. Asi se evita el reprocesamiento.
2. Entity Framework Query Optimization

El disable de entity framework para que sea más rápido.

public SimpleInventarioDao(Func<SimpleInventarioDbContext> contextFactory)

        {

            // performance optimizer.

            this.contextFactory = () =>

            {

                var context = contextFactory.Invoke();

                context.Configuration.AutoDetectChangesEnabled = false;

                return context;

            };

        }

Machine generated alternative text:
PROJECT 
ocøuG 
TEAM 
TEST 
ANALYZE 
public MerpContext 
public Database() 
this . Context 
Context get; private set; 
new Merpcontext(); 
this .Context. Configuration.AutoDetectChangesEnab1ed false; 
*region IDatabase Members 
public IQueryablecActivityType) ActivityTypes 
get return Context. ActivityTypes; 
public IQueryabIe«BusinesslJnit) BusinesslJnits 
get return Context. BusinessUnits; ) 
31:31 / 

Message Receiver.

Ahora se pueden ver como dos transcciones de vida larga en la base de datos a los procesos que miran las tablas del bus:



SOBRE PORQUE NO USO BUS EN MEMORIA – 3 RAZONES

1. Se pierde la capacidad de programar tareas (poniendo fecha de procesamiento del mensaje en el bus).
2. Se pierde la capacidad de ejecutar largos procesos de análisis de eventos. Si fuera en memoria se tendría que ejecutar todo primero antes de mostrarle cosas al usuario. Esto quizás si no fuera muy largo sería rapidísimo.
3. Es más difícil el multiusuario (cada vez que se hace commit ya puede un usuario ganar un turno en el event store)

Testing - EN EL CONNECTION STRING DEL BUS EL POOLING DEBE SER FALSE: Funciona

TEST: Soporta 1000 usuarios en simultaneo sobre el mismo aggregate!