20210222

segunda-feira, 22 de fevereiro de 2021 20:19

Vamos aos fatos, ela trabalha com organização de serviços do modelo (core) que estou chamando de processo, engessa eles de modo que realmente façam apenas uma coisa (no caso uso de método "executar" , por que como o código pode ser alterado , significa que mesmo assim pode cagar a ideia), esta parecido a um command e a pre-definição dos processos , plano de execução , para atender um objetivo/feature, esta parecido ao mediator.

Esses processos tem a premissa de servirem a apenas um objetivo, não guardarem estado (stateless), modificarem ou não o agregado passado com base na implementação do agregado (um serviço executa a modificação disponível no agregado) e deixar seguir o plano de execução.

Tá ai outra coisa que ficou legal, o plano de execução deve atender ao objetivo do agregado , logo, terei uma simplificação de pensar na modelagem pois : monto um plano de execução com base no agregado, esse plano de execução pode ter outros planos de execução, mas que compõem o principal para chegar ao mesmo objetivo.

O plano de execução pode utilizar diversos tipos de processo, como validação de politica para os dados de entrada ou dados gerados na saída, chamada a recurso externo, chamada a recurso interno, envio de e-mail, envio a filas, obtenção de dados externos, obtenção de dados internos , enfim, uma infinidade de coisas. Esses processos podem ou não alterar o estado do agregado contudo eles fazem apenas a alteração que o agregado já permite (implementa) então estão amarrados. Apenas o agregado pode alterar seu estado.

~~Apenas~~ o plano de execucao tem responsabilidade de persistir os dados gerados , logo, esse armazenamento ocorre sempre quando o processo termina de executar, caso de erro no armazenamento do artefato gerado pelo processo, é gerado o relatorio e armazenado com o erro e os dado ate dar o erro , caso o erro seja critico e nao seja possivel armazenar esses dados , seja por falha na comunicacao com servico externo , falha na persistencia, falha no codigo em si, o plano de execucao eh armazenado em arquivo ou de alguma maneira que n haja a possibilidade(minimo) de ocasionar erro, com a identificacao do plano , dados de entrada, operador executante , processos planejados, processo que deu erro ou erro geral. ~~Esse eh o plano DM por enquanto tem que ser impossivel de dar erro.~~ Essa estrutura de execucao tem o objetivo de facilitar a recuperacao de falhas. Com esse plano armazenado, assim que identificado e corrigido o erro, pode ser , caso necessario , rreprocessado o plano a partir do ultimo processo executado.

Reprocessamento do plano de execução:

- Deve criar novo identificador
- Adiciona o motivo do reprocessamento bem como a referência do anterior (que deu erro). Pensar sobre isso.
- Executa o plano normalmente com os mesmos dados de entrada com o nome do novo operador responsável pela execução , caso tenha mudado.

Então temos o seguinte: com base no requisito, defino um agregado , que se utilizara de entidades e objetos de valor em sua composição, seus motivos de mudança serão contemplados com a ajuda dos processos, que serão nomeados de acordo com essa necessidade, o agrupamento desses processos para atender um recurso (objetivo principal) do agregado, chama-se plano de execução, ele contem todos os processos necessários em ordem sequencial de execução para atender o objetivo único e bem definido (pensar em execucao em thread mas ai tem a questao da dependencia entre eles)

Essa modelagem deve ser refletida no frontend, de modo que eu tenha toda a implementação

testável separada do código de estruturação de tela (html por exemplo).
Sendo assim consigo testar isso de forma independente, sem necessariamente testar UI o tempo todo. Então conseguimos aplicar a separação de conceitos. Facilitara o teste, reuso, e implementação.

Medo: por ser particionadas as responsabilidades em cada um faz uma coisa, pode ter aumento violento da complexidade e facilidade de implementações ruins , logo n atendendo aomobjetico que eh facilidade de manutencao quando o projeto estiver maduro.

Objetivos principais da estrutura:

"Coisas a testar e que devem ser atendidas: fácil leitura , fácil depuração , fácil entendimento, fácil reuso , fácil tradução para outra linguagem de programação, ser facil para implementar testes de unidade e integração de processos de forma independente (se isso for possível já caracteriza a modelagem como foda) , fácil documentação viva, APIs independentes, acoplamento mínimo, composição bem definida , fácil adaptabilidade (resiliência), fácil atualização (isso é possível pois não será utilizada nenhuma dependência de terceiros no modelo), independência total de persistência, cache e fila (nenhum deles deve interferir na modelagem serão decididos por último) bem como permitir o uso de N tipos diferentes desses recursos (tecnologias etc), os processos podem ser separados em microsservicos e/ou utilizados como monólitos pois a comunicação será adaptativa por contratos e com uso de identificadores globais, será transparente ao procurar algum plano de execução feito por alguém e/ou pelo artefato gerado, ou seja , rastreabilidade total e fácil , para que seja possível identificar rapidamente falhas e desempenho ruim".

Validar modelagem em python, rust, go , c# e es6 (js) (denojs). Tem que funcionar apenas com a tradução, respeitando as maneiras de cada linguagem tratar algumas coisas, mas que não mudem o objetivo, resultado.

O que esta faltando eu definir e a questão do objeto "dado" que e passado no meio do fluxo, pois se eu considerar que montarei o objeto de entrada com os dados de cada processo, significa que eu preciso ter algo que e mutável mas que o controle de mudanças fique apenas nesse objeto, sendo assim estou, trafegando um agregado e obrigando aos processos a tarefa de executarem coisas que ajudem o agregado a executar esses comportamentos de mudança. Estava faltando ne, pois acabei de definir... será um agregado e apenas isso. Será prudente fazer uma interface para representar essa raiz de agregado para engessar.

Independente da base de persistência, temos que avaliar se e prudente gerar um hash para validar a autenticidade dos dados que forem passados para filas, persistência e afins, de modo que na comunicação do contexto A com o B , caso necessário o B pode consultar a autenticidade de um conjunto de dados gerados por um plano de execução.

Essa ideia vale também para a auditoria de dados e auditoria de uso .

Basicamente inicialmente o rastreio se dará pelo código gerado no plano de execução, então para saber o que aconteceu num plano de recurso A precisamos apenas informar seu código de identificação global.

Eu poderia persistir os dados criptografados , porém isso acarretaria num maior uso de CPU, então tem que calcular ate que ponto e viável ter esse recurso. Também adicionaria um problema caso seja utilizada uma ferramenta de BI que conecta direto no banco. Seria necessário criar a interface para isso e para o desenvolvedor consultar os dados, então teríamos de adicional para vários usos:

- Aplicação para consulta de dados persistidos criptografados que os exibe em clear-text para pessoas com permissão.
- Interface para intermediar acesso aos dados persistidos criptografados para utilizar em aplicação BI (avaliar se funciona com power Bi por exemplo. se é possível)
- Aplicação para auditoria de dados (validação de hashes e integridade de persistencia)

- Aplicação para auditoria de utilização (no caso de quem executou, mudança de dados etc, "todos"(a definir) os "movimentos" do operador).
- Aplicação para auditoria de erros.
- Aplicação para auditoria de desempenho.
- Aplicação para visualizar qualquer plano de execução que foi executado.
- Interface para permitir validação de consistência de dado , valida o hash em relação ao conteúdo. Se isso for feito deve ter seu uso muuuito bem rastreado.
- Aplicação para reprocessamento de plano: entro com o plano que deu problema -> exibe o plano problemático na tela -> sistema carrega e associa os dados -> executo o reprocessamento

Essas aplicações de auditoria podem exibir os planos de execução pais da mesma maneira:

- Exibir mesma sequencia de execução
- Operador que executou
- Dados gerados em cada processo (vide autorização de visualização aberta?)
- Validações (regras e politicas) utilizadas em cada processo
- Todos os tipos de relatório gerado no plano de execução, separados por processo dentro do plano
- Desempenho geral do plano de execução
- Alocação de dados em memoria (quantidades respectivas)
- Consumo de CPU
- Desempenho de cada processo dentro do plano

Tudo deve exibir dados de identificação do plano de execução (versão, nome, id global), data de inicio de execução, data fim de execução, identificação do processo (versão, nome), data inicio execução, data fim execução.
Obs.: essas datas de execução devem ser registradas independente de erro.

Criar log local em arquivo como plano DM de rastreio. DM = deu merda.

Para teste , independente do ambiente, ao autenticar com usuário de perfil QA ou DEVTESTE, os registros gerados por ele em qualquer plano de execução são rotulados com a palavra QA ou DEVTESTE.. Fica fácil eliminar dados de teste se necessário (lixo no geral) em qualquer ambiente.