RELATÓRIO DE ANÁLISE DE DESENVOLVIMENTO

PROJETO E-COMMERCE -AVALIAÇÃO AP004

TIME: 05

DATA : 24/05/2024

EQUIPE E PAPÉIS

Alan Carlos - Líder Técnico
Breno Rios - Desenvolvedor
Ezequiel Lobo - Desenvolvedor
Marcelo da Silva - Desenvolvedor

JUSTIFICATIVA DE ALTERAÇÕES EM MÓDULOS

1. Alteração da Arquitetura Monolítica para Micro Serviços.

    1.1. **Escalabilidade**: Cada micro serviço pode ser escalado de forma independente conforme a demanda. Isso acaba se tornando mais eficiente e econômico, pois permite alocar recursos específicos para os serviços que realmente precisam.

    1.2. **Desenvolvimento Paralelo**: Diferentes equipes podem trabalhar simultaneamente em diferentes serviços, sendo assim, acelerando o tempo de desenvolvimento e implantação.

    1.3. **Implantação Contínua**: Os microsserviços facilitam práticas de DevOps, como a integração e entrega contínua (CI/CD), isso permite com que realizem atualizações frequentes e mais seguras sem afetar o sistema inteiro.

    1.4. **Manutenção Simplificada**: Bugs podem ser isolados e corrigidos em serviços específicos sem necessidade de fazer deploy do sistema inteiro.

    1.5. **Isolamento de Falhas**: Problemas em um micro serviço não necessariamente afetam os outros serviços, aumentando a resiliência do sistema.

2. Utilização da biblioteca Serilog para obtenção de logs do sistema.

    2.1. **Melhor análise**: Logs estruturados podem ser facilmente consultados e filtrados, ajudando na rápida identificação de problemas e na análise de padrões de comportamento da aplicação.

    2.2. **Compatibilidade**: o serilog integra-se bem com ferramentas de monitoramento e dashboards, facilitando a visualização e análise dos logs.

    2.3. **Auditoria**: Facilita a criação de logs de auditoria detalhados, úteis para a conformidade com normas de segurança e regulamentações.

3. Utilização dos Princípios SOLID.

    3.1. **Responsabilidades Claras**: Cada classe tem uma única responsabilidade, o que torna o código mais fácil de entender e modificar.

    3.2. **Confiabilidade**: Assegura que a herança é usada corretamente, mantendo a substituibilidade e coesão do sistema.

    3.3. **Crescimento Sustentável**: A aplicação dos princípios SOLID resulta em um código base que pode crescer e se adaptar com o tempo sem se tornar insustentável.

4. Implementação de testes.

    4.1. **Detecção Precoce de Erros**: Testes ajudam a identificar e corrigir bugs nas fases iniciais do desenvolvimento, antes que se tornem problemas maiores.

    4.2. **Facilidade de Refatoração**: Testes asseguram que mudanças no código, como refatorações, não introduzem novos bugs, proporcionando uma base sólida para melhorias contínuas.

    4.3. **Estabilidade do Sistema**: Testes contínuos garantem que o sistema mantenha seu comportamento esperado ao longo do tempo, aumentando a confiança na estabilidade da aplicação.

    4.4. **Redução de custos de correção**: Identificar e corrigir bugs nas fases iniciais do desenvolvimento é geralmente mais barato do que fazer isso em produção.

5. Realização de documentação do sistema.

    5.1. **Referência Detalhada**: A documentação fornece uma referência detalhada sobre a arquitetura, componentes e funcionamento do sistema, facilitando a manutenção e atualização do código.

    5.2. **Curva de Aprendizado Reduzida**: Novos membros da equipe podem se familiarizar rapidamente com o sistema lendo a documentação, reduzindo o tempo necessário para se tornarem produtivos.

    5.3. **Orientação e Suporte**: A documentação oferece orientação sobre práticas recomendadas e padrões de codificação adotados no projeto.

6. Utilização do git flow.

    6.1. **Padronização**: Estabelece convenções de nomenclatura e uso de branches, promovendo consistência e facilitando a compreensão do fluxo de trabalho por toda a equipe.

    6.2. **Colaboração Facilitada**: O Modelo de ramificação promove um ambiente colaborativo onde múltiplos desenvolvedores podem trabalhar em paralelo em diferentes funcionalidades ou correções.

6.3. **Histórico Estruturado**: Mantém um histórico de commits bem organizado, facilitando a auditoria e a compreensão das mudanças ao longo do tempo.

7. Utilização de mensageria

7.1. **Processamento assíncrono**: Mensageria é ideal para cenários onde o processamento assíncrono é vantajoso, como processamento de pedidos em segundo plano, envio de e-mails ou notificações push.

Exemplo: Quando um cliente faz um pedido, em vez de processar imediatamente todas as etapas (verificação de estoque,, geração de faturas, etc.) de forma síncrona, você pode usar mensageria para enviar uma mensagem para uma fila de pedidos. Isso permite que o sistema processe pedidos em segundo plano, sem atrasar a resposta ao cliente.

8. Utilização de ferramenta para validação de dados de entrada

8.1. **FluentValidation:** Você pode definir a regra de utilização apenas uma vez e reutilizar em diferentes partes do seu software

RELATÓRIO DE ATIVIDADES

TAREFA	RESPONSÁVEL	TEMPO PREVISTO	TEMPO EXECUTADO
Alteração da Arquitetura Monolítica para micro serviços	Toda equipe	1 mês	
Utilização da biblioteca Serilog	Alan Santos	3 dias	
Implementação de testes na aplicação	Breno Rios	1 semana	
Documentação do sistema	Ezequiel Lobo	2 semanas	
Utilização do GitFlow	Toda equipe	Em todo desenvolvimento da aplicação	
Implementar ferramenta de validação de entrada de dados	Marcelo da Silva	3 dias	