ARQUITETURA DE SOFTWARE

João Choma Neto
joao.choma@unicesumar.edu.br
https://github.com/JoaoChoma/arquitetura-software
Unicesumar – Maringá

ARQUITETURA DE MICROSERVIÇOS

MICROSERVIÇOS

- Microserviços são uma abordagem arquitetural no desenvolvimento de software onde uma aplicação é estruturada como um conjunto de pequenos serviços
- Cada serviço executa uma única função

MICROSERVIÇOS

- Cada serviços comunica-se com outros serviços através de interfaces bem definidas, geralmente APIs.
- Cada microserviço é desenvolvido, implantado e escalado independentemente, permitindo uma maior flexibilidade e modularidade na gestão do sistema.

Características dos Microserviços

- **Descentralização**:
  - Cada microserviço é responsável por uma parte específica da funcionalidade da aplicação, e pode ser desenvolvido e mantido por diferentes equipes.

Características dos Microserviços

- **Independência de Implantação**:
  - Os microserviços podem ser implantados, atualizados, escalados e reiniciados de forma independente, sem afetar os demais serviços.

Características dos Microserviços

- **Escalabilidade**:
  - Permite escalar apenas os serviços que necessitam de mais recursos, ao invés de escalar toda a aplicação.

Características dos Microserviços

- **Tolerância a Falhas:**
  - A falha em um microserviço não necessariamente compromete toda a aplicação, pois os outros serviços continuam funcionando.

Características dos Microserviços

- **Tecnologia Heterogênea:**
  - Cada microserviço pode ser desenvolvido usando diferentes tecnologias e linguagens de programação, desde que respeitem as interfaces de comunicação.

Benefícios dos microserviços

- Manutenção Facilitada: Código mais simples e específico facilita a correção de bugs e a implementação de novas funcionalidades.
- Implantação Contínua: Permite implantar novas versões de serviços individualmente, facilitando a integração contínua e entrega contínua (CI/CD).

Cuidados com a utilização

- Gerenciamento de Dados: Cada serviço pode ter seu próprio banco de dados, necessitando de uma abordagem consistente para transações de dados distribuídas.
- Segurança: Aumenta a superfície de ataque, requerendo uma atenção especial à segurança de cada serviço e suas comunicações.

e-COMMERCE

EM MICROSERVIÇOS

SERVIÇO DE AUTENTICAÇÃO

- Responsável por gerenciar o login, logout e a criação de usuários.
- Verifica as credenciais e emite tokens de autenticação.

SERVIÇO DE CATÁLOGO DE PRODUTOS

- Gerencia os dados dos produtos disponíveis para venda.
- Permite adicionar, atualizar, remover e listar produtos.

SERVIÇO DE CARRINHO DE COMPRAS

- Gerencia o carrinho de compras do usuário.
- Adiciona, remove e lista itens no carrinho.

SERVIÇO DE PEDIDO

- Processa pedidos de compra.
- Gerencia o status do pedido e histórico de compras.

SERVIÇO DE PAGAMENTO

- Processa pagamentos e transações financeiras.
- Integra com gateways de pagamento.

SERVIÇO DE NOTIFICAÇÃO

- Envia notificações por email ou SMS sobre o status do pedido, promoções, etc.

PRODUTOS

Firebase

- **Firebase Authentication**: Serviço de autenticação para gerenciar usuários.
- **Firestore**: Banco de dados NoSQL para armazenamento de dados.
- **Firebase Cloud Messaging (FCM)**: Serviço de envio de notificações push.

AWS (Amazon Web Services)

- **AWS Lambda**: Serviço de computação sem servidor para executar código em resposta a eventos.
- **Amazon S3**: Armazenamento de objetos para arquivos e dados.
- **Amazon RDS**: Banco de dados relacional gerenciado.
- **Amazon SNS**: Serviço de notificações simples para envio de mensagens.

Azure (Microsoft)

- **Azure Functions**: Serviço de computação sem servidor para executar código sob demanda.
- **Azure Cosmos DB**: Banco de dados NoSQL globalmente distribuído.
- **Azure Logic Apps**: Serviço de automação e orquestração de fluxos de trabalho.

Google Cloud Platform (GCP)

- **Google Cloud Functions**: Serviço de computação sem servidor.
- **Google Cloud Pub/Sub**: Serviço de mensageria para troca de mensagens assíncronas.
- **Google Cloud Firestore**: Banco de dados NoSQL em tempo real.

Heroku

Heroku Postgres: Banco de dados PostgreSQL gerenciado.

Heroku Redis: Armazenamento em cache e banco de dados de chave-valor.

Heroku Connect: Sincronização de dados entre Salesforce e Heroku Postgres.

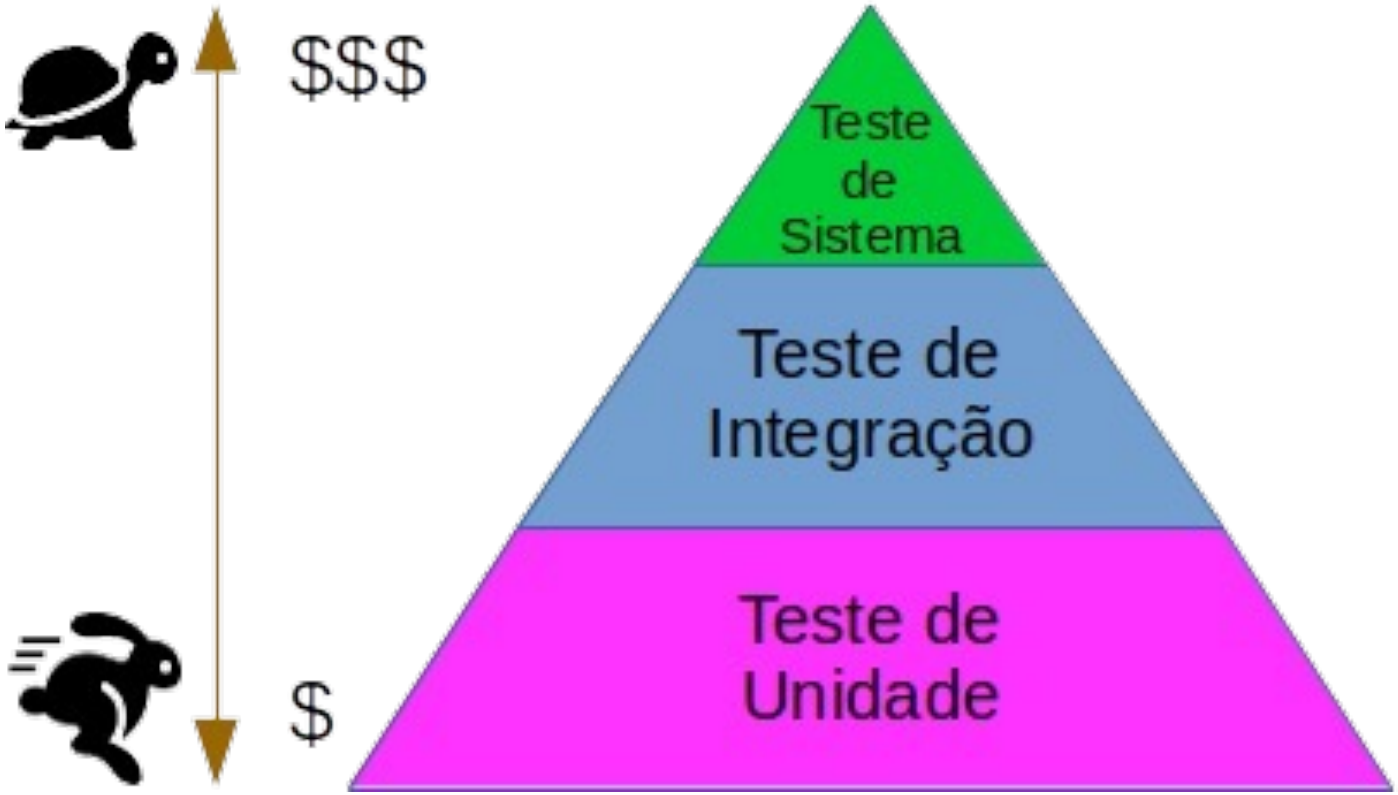
Modelagem de uma arquitetura

- **Autenticação e Autorização**: Firebase Authentication, Auth0
- **Armazenamento de Dados**: Firestore, Amazon RDS, Azure Cosmos DB
- **Computação Sem Servidor**: AWS Lambda, Azure Functions, Google Cloud Functions
- **Mensageria e Notificações**: Firebase Cloud Messaging, Amazon SNS

Ideia

1. **Autenticação**: Usar Firebase Authentication para gerenciar os logins e a criação de contas.
2. **Banco de Dados**: Usar Firestore para armazenar dados de produtos, usuários, e pedidos.
3. **Funções de Backend**: Usar AWS Lambda para processar eventos como a criação de pedidos e a verificação de pagamentos.
4. **Mensageria**: Usar Google Cloud Pub/Sub para enviar mensagens entre serviços, como notificações de novos pedidos.
5. **Notificações**: Usar Firebase Cloud Messaging para enviar notificações push aos usuários sobre o status do pedido.

TESTE DE UNIDADE


$$$
$
Teste de Sistema
Teste de Integração
Teste de Unidade
Maior granularidade
Menor quantidade
Mais lentos
Maior custo
Menor granularidade
Maior quantidade
Mais rápidos
Menor custo

TESTE UNITÁRIO

- Teste unitário é uma prática de teste de software na qual cada componente individual ou "unidade”

TESTE UNITÁRIO

- Uma unidade pode ser a menor parte testável de um aplicativo
- Função
- Método
- Classe
- Módulo

TESTE UNITÁRIO

- O objetivo principal dos testes unitários é identificar defeitos nas unidades de código antes que elas sejam integradas ao restante do sistema

TESTE UNITÁRIO

- Isolar as unidades de código e testá-las independentemente
- Os desenvolvedores podem verificar se cada unidade executa as tarefas designadas de acordo com suas especificações

TESTE UNITÁRIO

- Identificar problemas cedo no processo de desenvolvimento
- Tornar mais fácil e mais econômico corrigir erros

TESTE UNITÁRIO

- Cada teste unitário deve ser automatizado para que possa ser executado repetidamente e integrado a fluxos de trabalho de desenvolvimento contínuo

TESTE UNITÁRIO

- Processo estruturado para garantir que cada componente individual do código seja testado de maneira isolada

UNIDADE

- **Identifique as Unidades de Código**
- Determine as unidades de código que você deseja testar

FRAMEWORK

- **Escolha um Framework de Testes**
- Utilize um framework de testes apropriado para a linguagem de programação que você está usando

FRAMEWORK

- JUnit (Java)
- NUnit (.NET)
- pytest (Python)
- Jest (JavaScript)

CASOS DE TESTE

- **Crie Casos de Teste**
- Para cada unidade de código, escreva casos de teste que cubram diferentes cenários
- Incluir casos em que a unidade deve produzir resultados corretos
- Incluir casos em que deve lidar com entradas inválidas ou inesperadas

TESTES

- **Escreva os Testes**
- Escreva o código dos testes usando o framework escolhido

TESTES

- **Dados de Teste**
- Inserir dados de teste relevantes para os casos de teste
- Criação de objetos fictícios
- Valores de entrada simulados
- Cenários de teste específicos

EXECUTAR

- **Execute os Testes**
- **Verifique os Resultados**
- O framework de testes geralmente fornecerá relatórios indicando quais testes passaram e quais falharam

REPETIR

- Repetir o processo de teste para todas as unidades de código
- Cada nova modificação recursos ao código repetir os testes
- Cada adição novos recursos, criar novos testes de unidade e repetir os testes

ARQUITETURA DE SOFTWARE

João Choma Neto
joao.choma@unicesumar.edu.br
https://github.com/JoaoChoma/arquitetura-software
Unicesumar – Maringá