Evolução da Arquitetura de sistemas WEB

Um contexto histórico

Entendendo a evolução das aplicações WEB

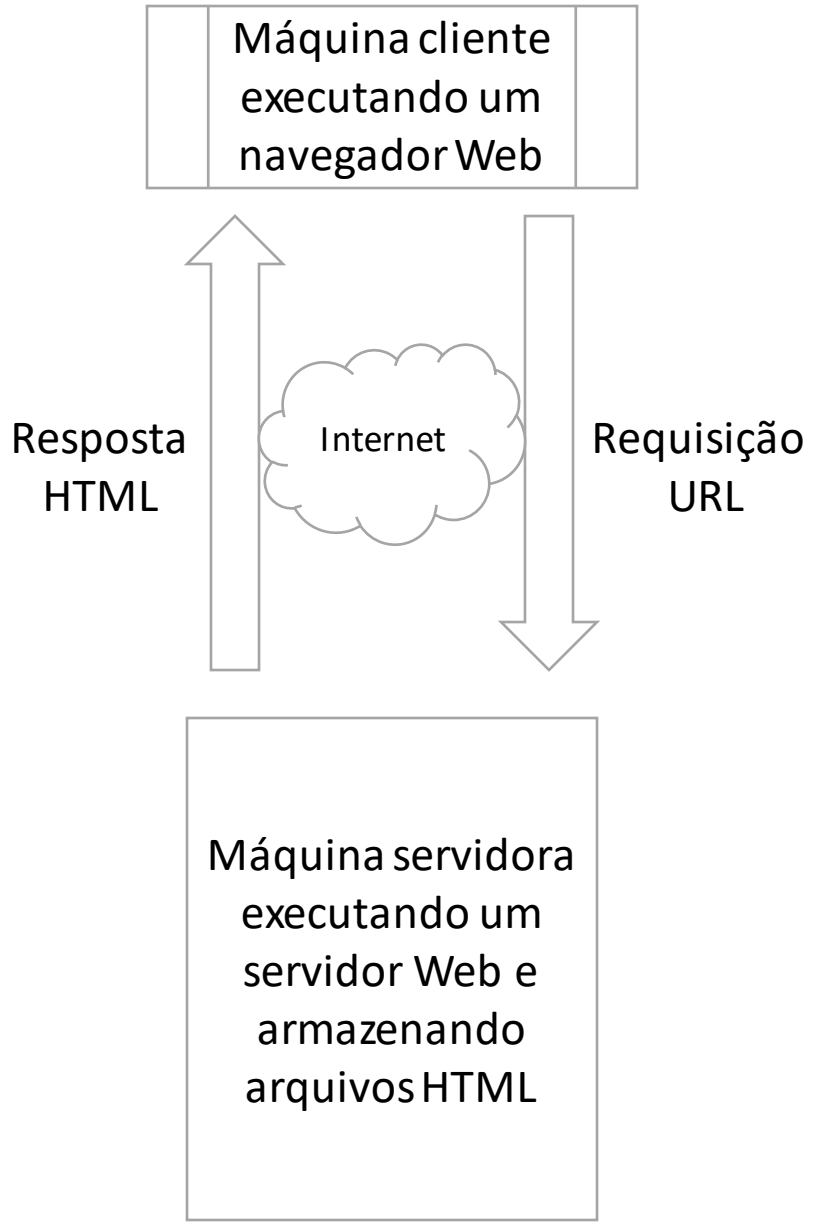
Vamos começar entendendo sobre como os sistemas Web evoluíram de um conjunto de páginas estáticas até aplicações complexas que se valem de uma arquitetura cliente servidor.

Evolução das aplicações WEB

- A internet como conhecemos surgiu em 1993 quando o CERN (Conseil Européen pour la Recherche Nucléaire) colocou sua tecnologia de “World Wide Web” em domínio público.
- Ainda no mesmo ano surgiu o “Mosaic 1.0” primeiro navegador WEB.
- Os primeiros servidores WEB hospedavam apenas páginas estáticas → Hipertextos acessáveis a partir de uma URL (Uniform Resource Locator) usando protocolo HTTP (Hypertext Transfer Protocol).

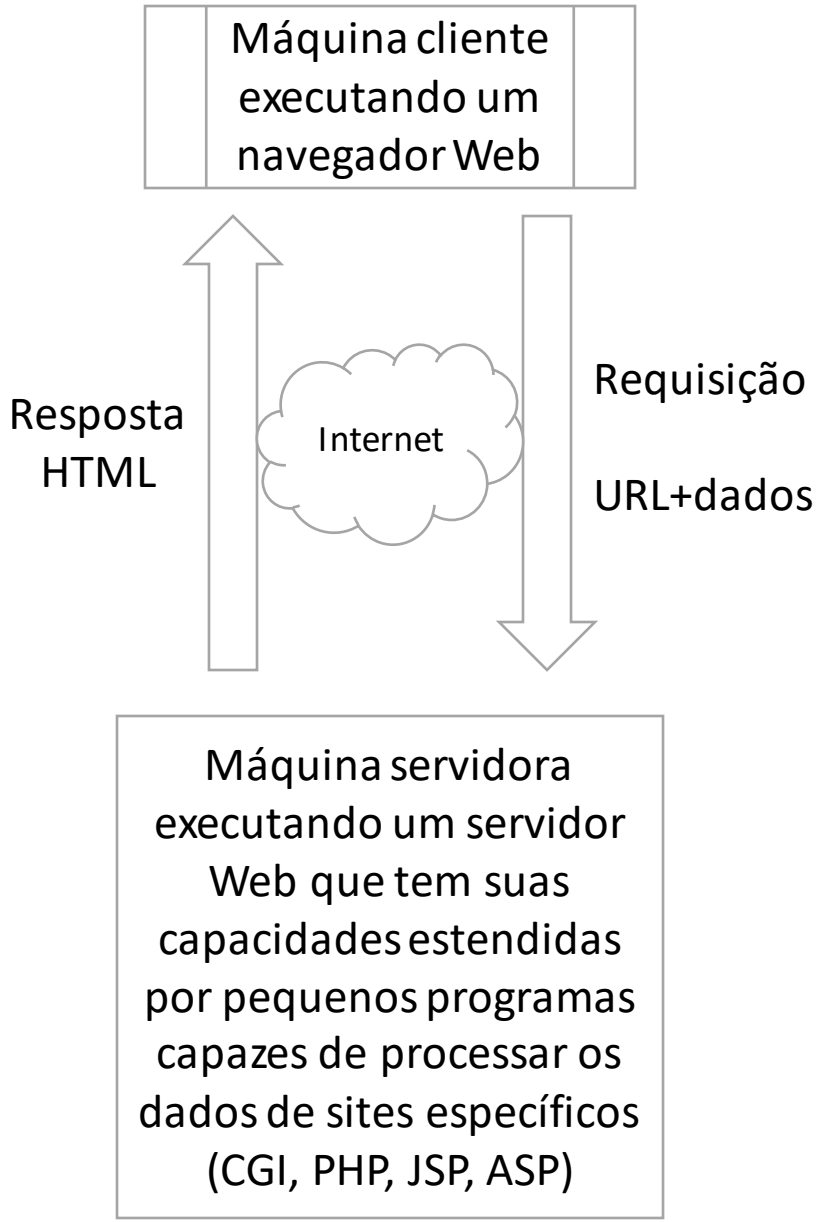
WebSite estático

- Navegador solicita arquivo HTML localizado em uma URL usando protocolo HTTP.
- O servidor Web que contém aquele recurso responde enviando o arquivo solicitado.
- O navegador “renderiza” o HTML recebido e exibe.



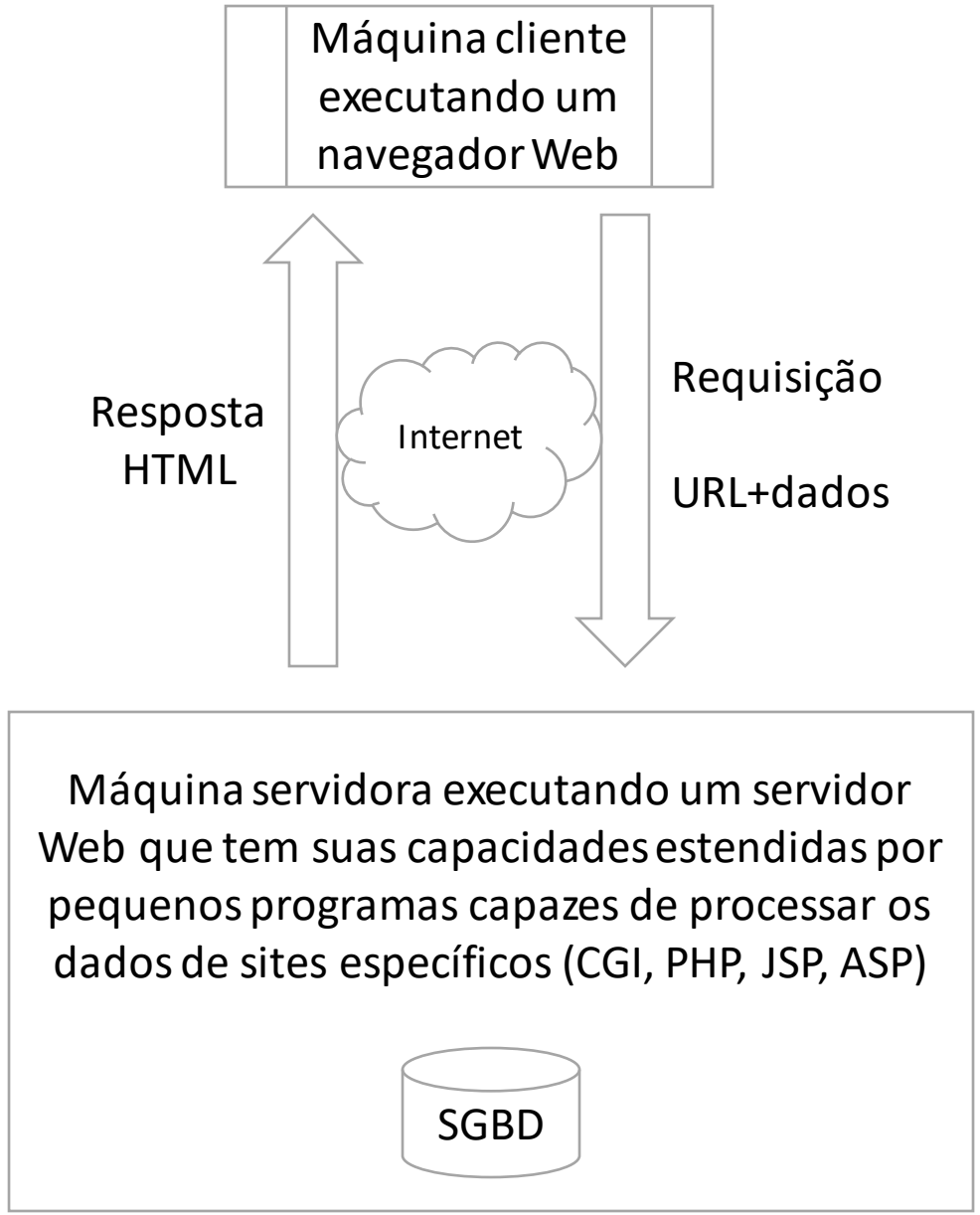
WebSite dinâmico (I)

- A solicitação HTTP do navegador compreende uma URL + dados.
- O servidor Web hospeda programas capazes de processar os dados e gerar páginas resposta dinamicamente (HTML).
- O navegador “renderiza” o HTML recebido e exibe.



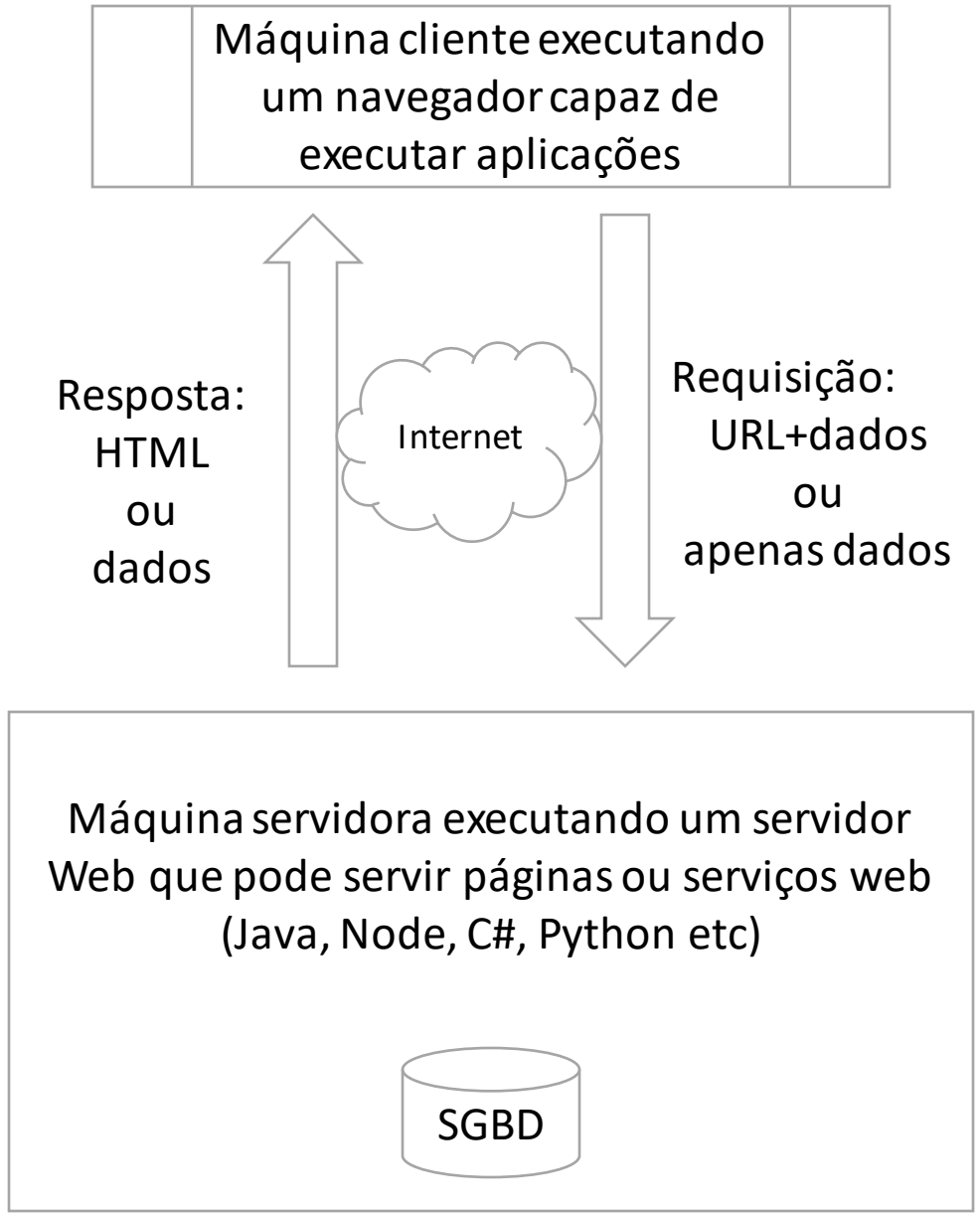
WebSite dinâmico (II)

- A medida que evoluíram estes “pequenos programas” passaram a ser capazes de acessar Sistemas Gerenciadores de Bancos de Dados (SGBDs)
- Os SGBDs podiam estar hospedados na mesma máquina servidora onde estava executando o servidor Web ou não.
- O navegador “renderiza” o HTML recebido e exibe.



Aplicação WEB

- O navegador não solicita apenas páginas. Pode enviar dados e receber dados.
- O navegador faz mais do que apenas renderizar HTML. Ele executa aplicações no lado do cliente (JavaScritp).
- O servidor é capaz de executar WebServices



Aplicação WEB: arquitetura cliente servidor

- Cliente
  - Client Side Rendering
    - Consomem uma API de serviço
    - A geração do HTML é feita no próprio cliente
    - JavaScript, React, Angular (MVC)
  - Server Side Rendering
    - Executa no servidor (devolve apenas HTML como nos sites dinâmicos) → performance
    - Pode consumir APIs de serviços
    - Next
- Servidor
  - Dois padrões de web services/APIS
    - Baseados em protocolos SOAP
      - Usam SOAP sobre HTTP ou outro protocolo
      - Usam XML e WSDL para a transferência de dados
    - Baseados em HTTP não SOAP
      - Usam REST ou qualquer outra coisa
      - Usam JSON para transferência de dados
      - Em termos de organização podem ser “monolitos” ou “micros serviços”
      - Em termos de implantação podem usar “containers” ou “serverless”.

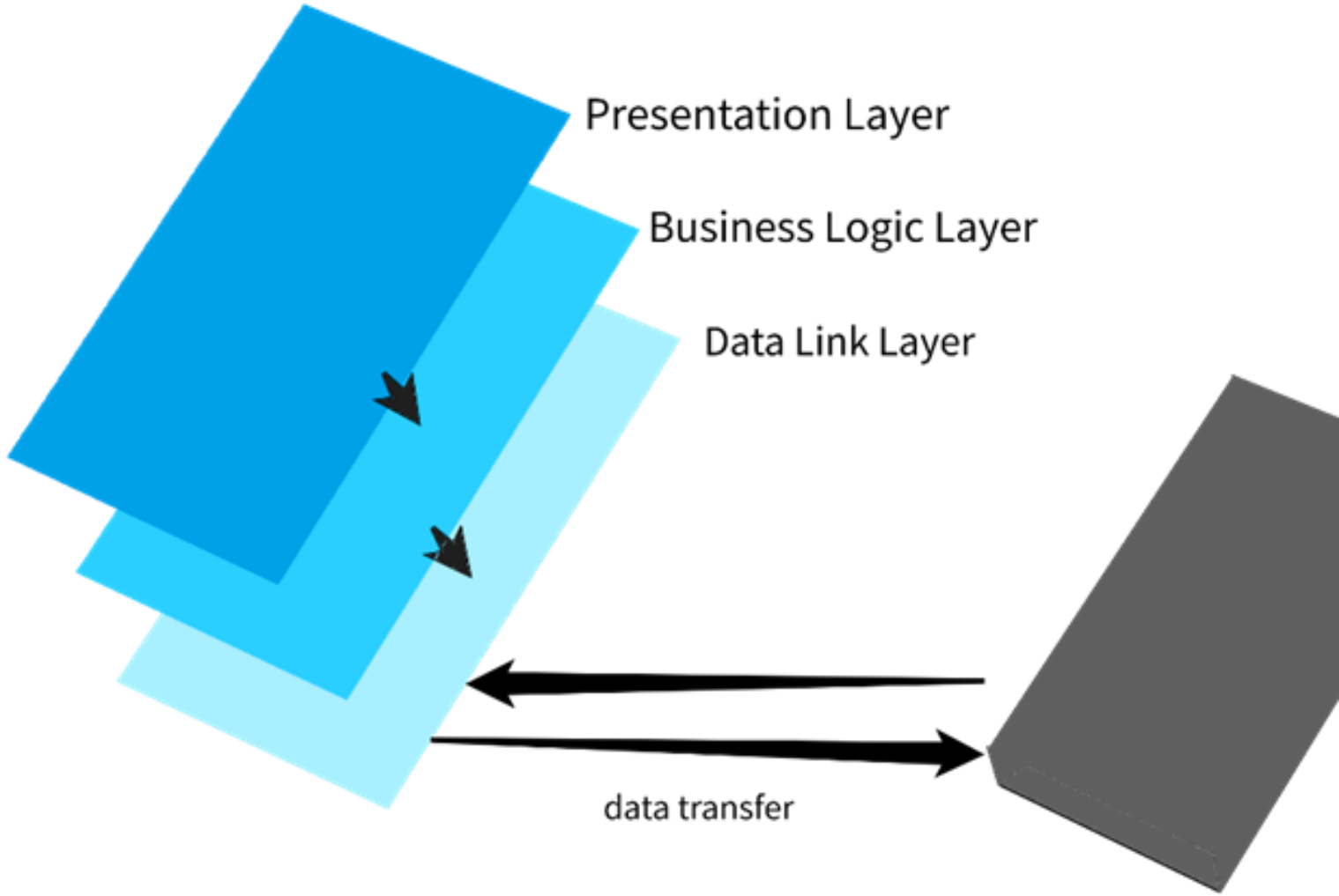
Evolução dos modelos arquiteturais das aplicações WEB

- Na medida que a tecnologia evoluiu os modelos arquiteturais tiveram de dar conta de aplicações que se tornaram, por um lado mais ricas, por outro mais difíceis de manter
- Surgem assim modelos arquiteturais robustos
  - Arquitetura em camadas lógicas
  - Arquitetura em camadas físicas
  - Arquitetura orientada a serviços
  - Arquitetura baseada em micros serviços

MARISTA

Arquitetura em camadas lógicas

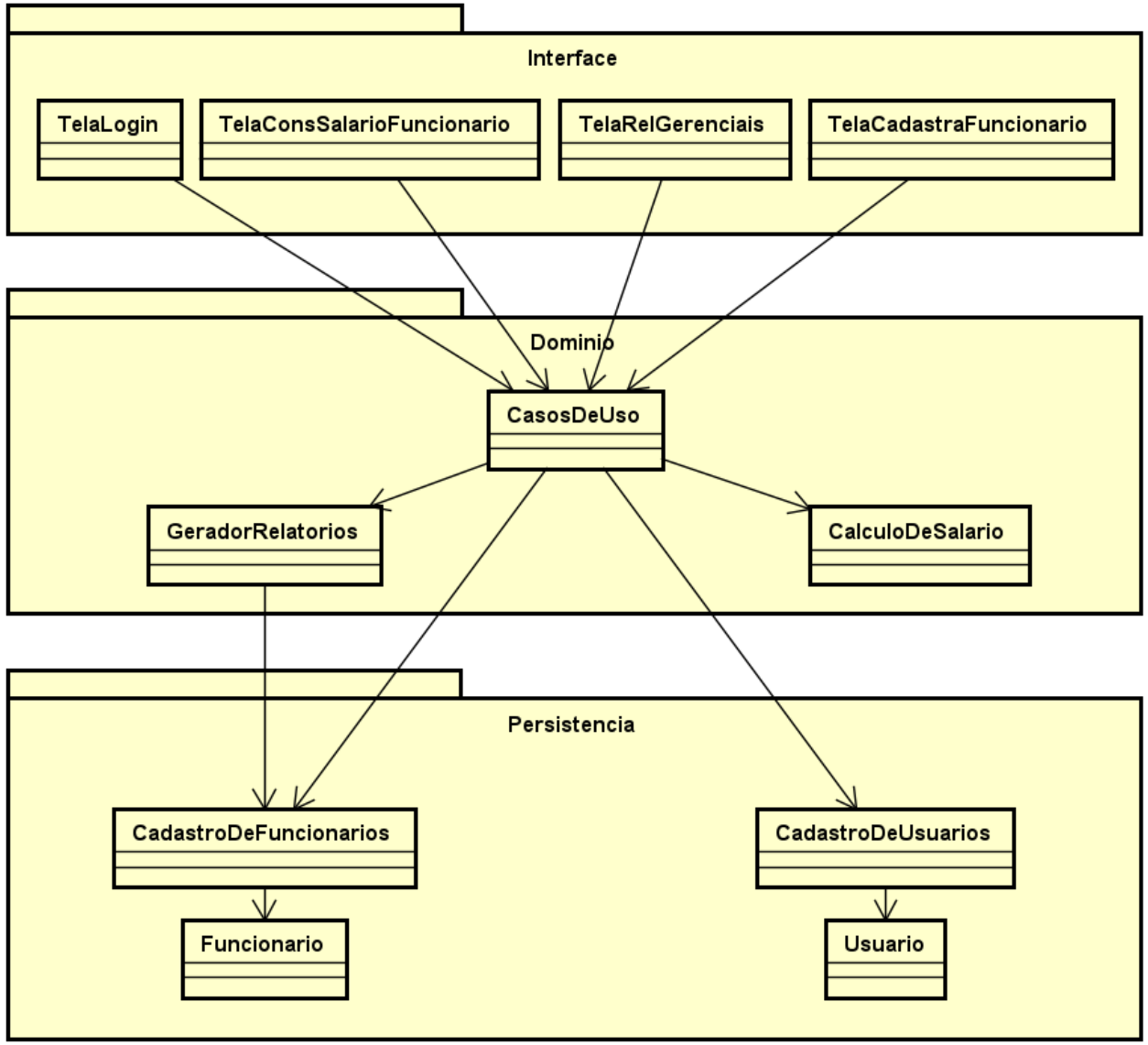
- Trabalha o princípio de separação de responsabilidades
- Cada camada fica dedicada a um tipo de responsabilidade
  - Apresentação
  - Negócio
  - Dados



Detalhamento das camadas

- Camada de apresentação (ou interface)
  - Mantém as classes que tratam da interação com o usuário
- Camada de negócio (ou domínio)
  - Mantém toda a lógica pertencente ao domínio. Deve estar completamente desvinculada a interface com o usuário
- Camada de persistência (ou acesso a dados)
  - Mantém as classe responsáveis pelo acesso aos dados da aplicação

Organização do sistema (monolito) em camadas lógicas

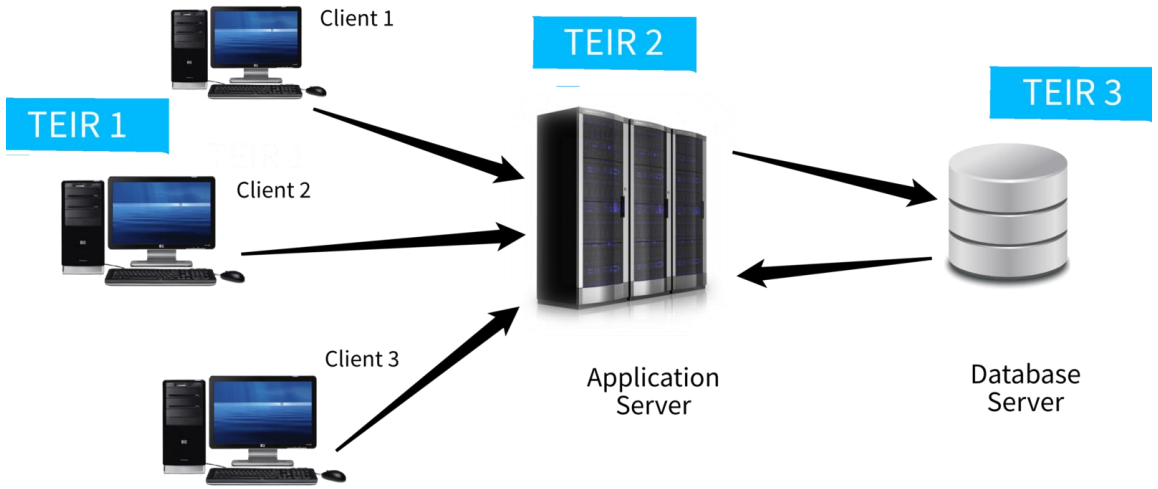


Vantagens e desvantagens

- Vantagens
  - Simples de implementar se comparada com outras abordagens
  - Oferece abstrações que permitem separar responsabilidades
  - O isolamento entre as camadas permite que alterações em uma não se reflitam nas outras
  - O software se torna mais fácil de gerenciar devido ao baixo acoplamento
- Desvantagens
  - Não oferece escalabilidade
  - Leva naturalmente para uma estrutura monolítica
  - O fluxo de dados de uma camada para outra por vezes é desnecessário

Arquitetura em camadas físicas

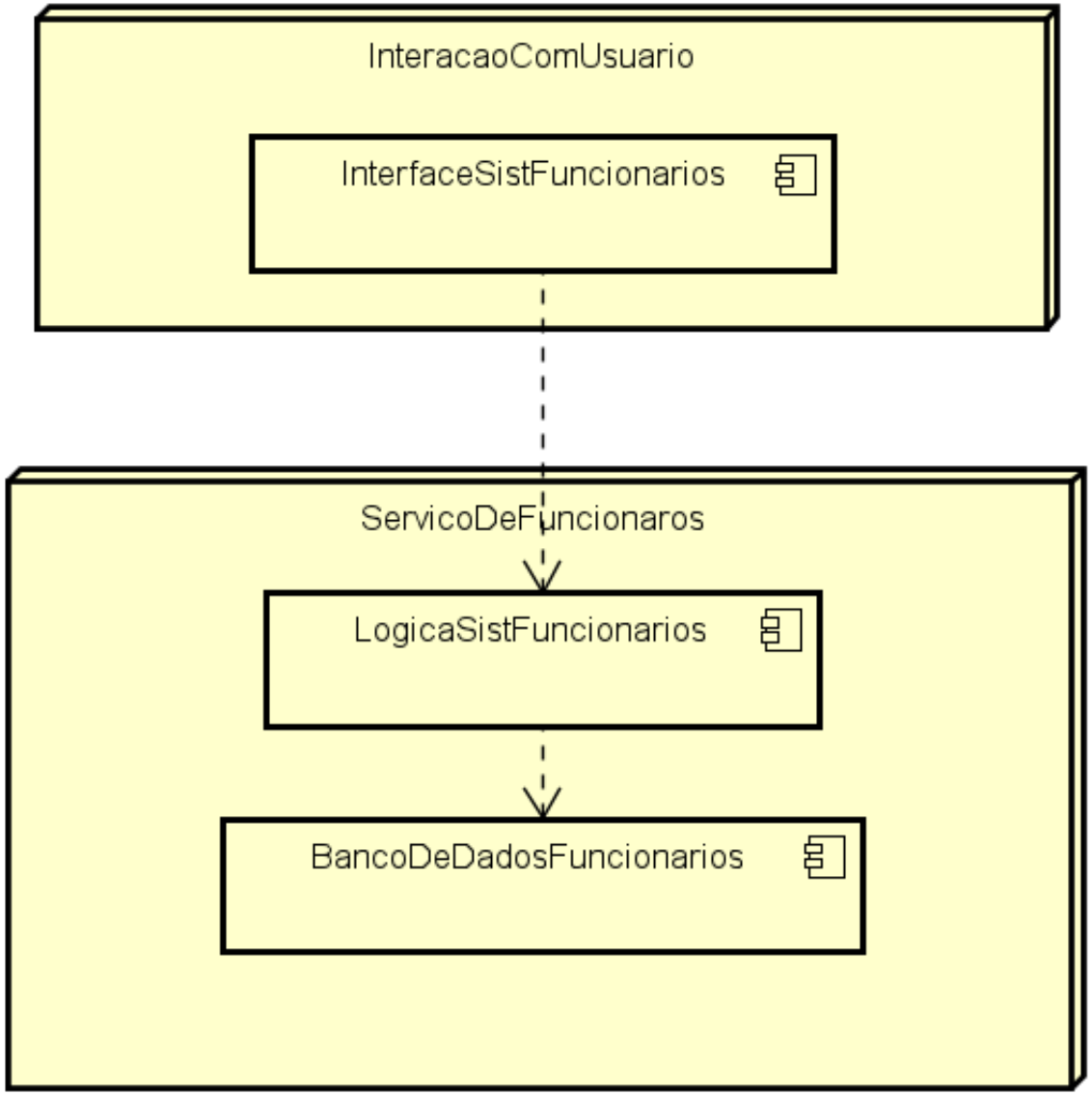
- Esta arquitetura divide o software em camadas físicas baseadas no princípio de comunicação “cliente-servidor”
- Este tipo de arquitetura pode ter uma ou mais camadas físicas separando as responsabilidades entre o provedor dos dados e o consumidor



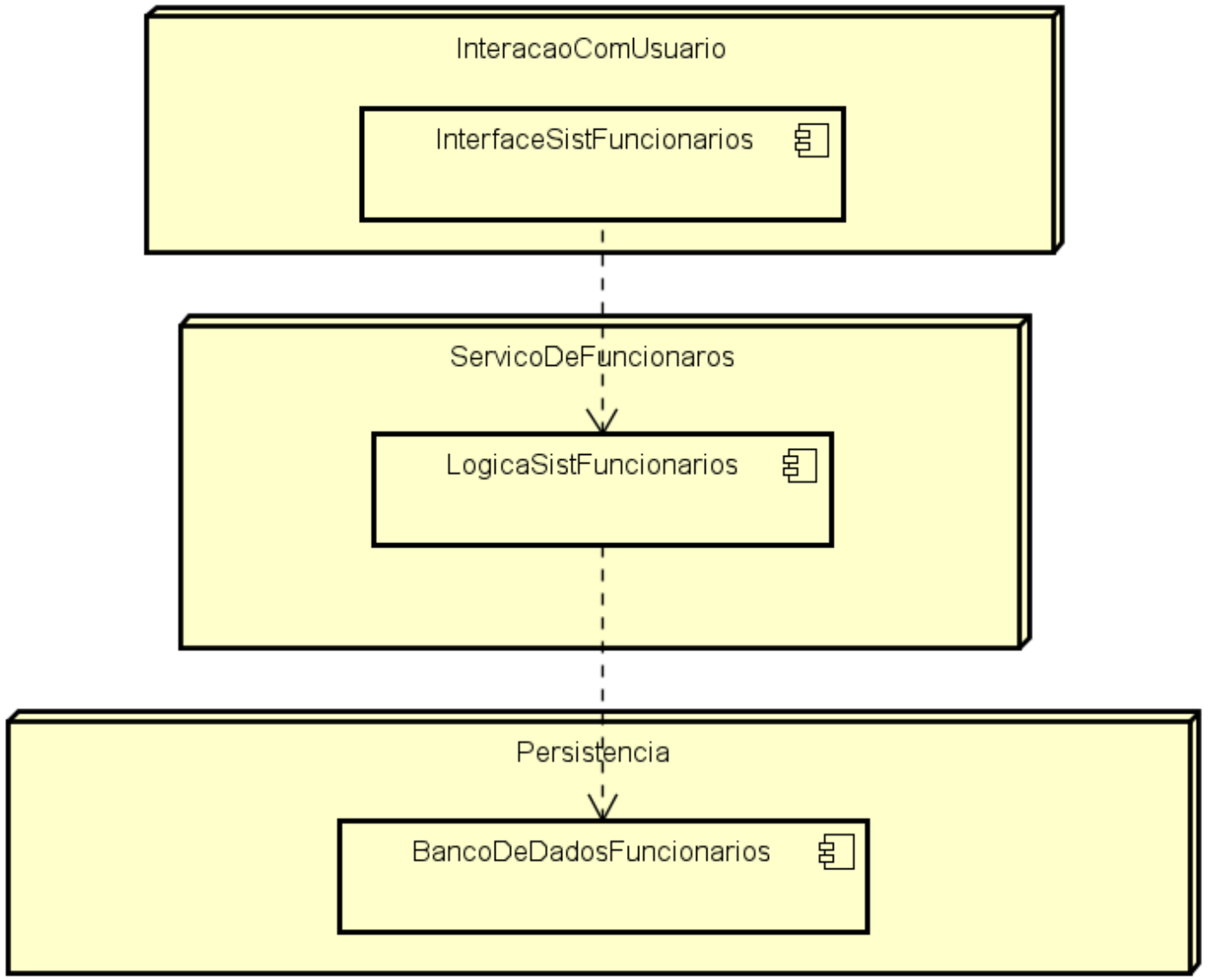
Características

- Utilizam um padrão de “requisição-resposta” para a comunicação entre as camadas físicas.
- Ao contrário do monolito organizado unicamente em camadas oferece escalabilidade tanto horizontal (mais nodos) como vertical (mais poder de processamento em um único nodo)
- Pode ser organizado em 2, 3 ou n camadas físicas

Organização do sistema em três camadas lógicas e duas camadas físicas



Organização do sistema em três camadas lógicas e três camadas físicas

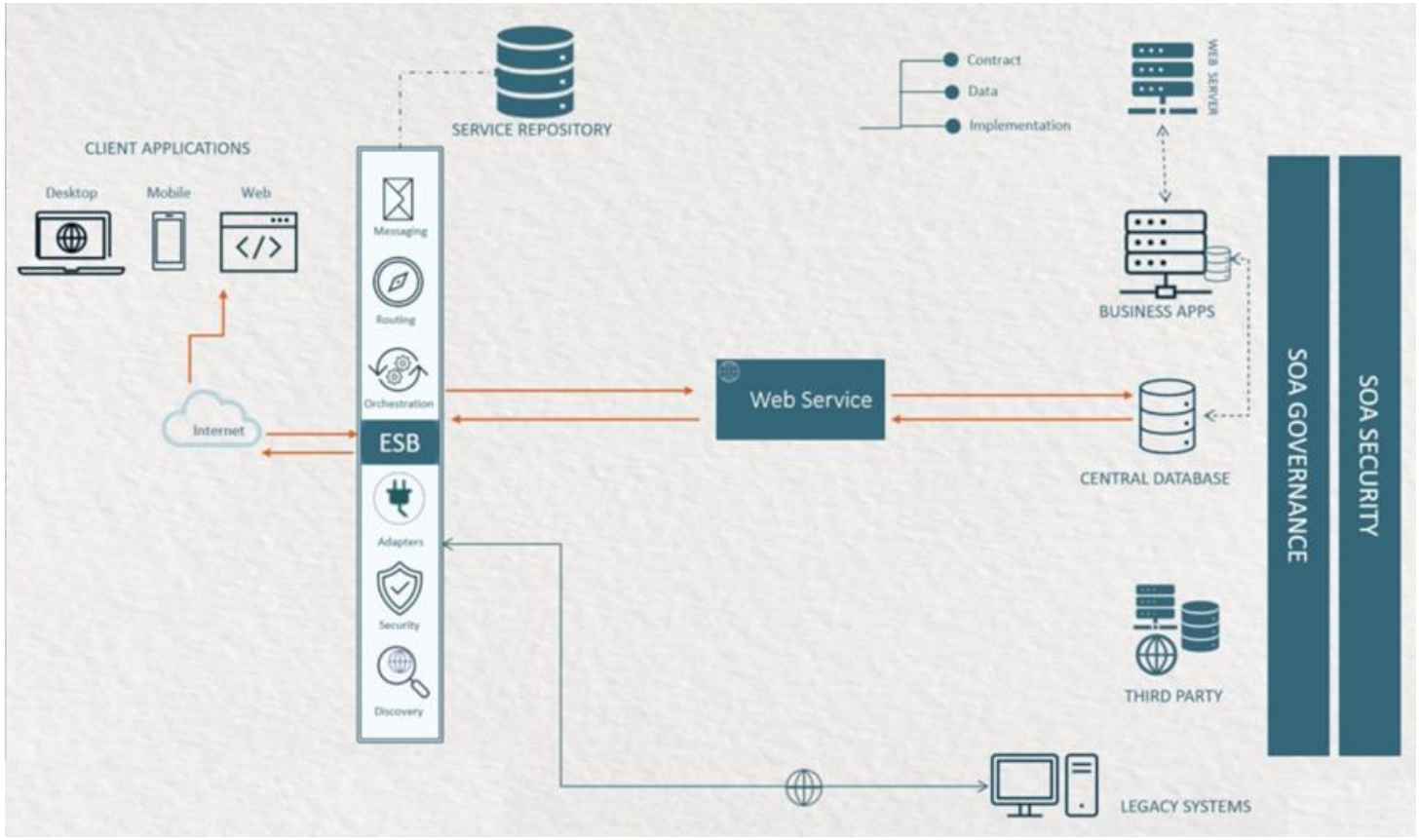


Arquitetura orientada a serviços

- Em uma arquitetura baseada em serviços (SOA) componentes e aplicações comunicam-se entre si usando serviços bem definidos
- Arquiteturas SOA compreendem 5 elementos:
  - Serviços
  - “Service Bus” (ESB)
  - Catalogo de serviços
  - Segurança
  - Governança

Detalhamento

- Requisições são enviadas usando um protocolo padrão
- Requisições são tratadas pelo ESB → orquestração e roteamento
- O ESB usa o catalogo de serviços para fazer o roteamento
- As requisições/respostas tem de estar de acordo com as regras de segurança e governança



Classificação e tipos

- Classificação
  - Serviços atômicos: provem funcionalidade que não podem decompostas
  - Serviços compostos: um agregado de vários serviços atômicos que fornecem uma funcionalidade mais complexa

- Tipos
  - Serviço de entidades
  - Serviços de domínio
  - Serviços utilitários
  - Serviços integrados
  - Serviços de aplicação
  - Servicos de segurança

Arquitetura de micros serviços

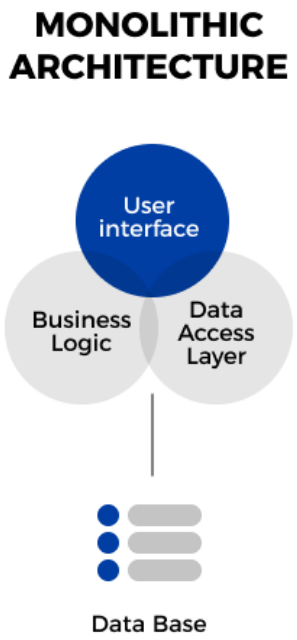
- Uma abordagem de desenvolvimento de uma única aplicação como um conjunto de pequenos serviços, cada um executando em seu próprio processo e se comunicando através de mecanismos leves (normalmente HTTP)
- São construídos ao redor de objetivos de negócio e podem ser instalados (deployed) de maneira independente e de forma automatizada
- O gerenciamento central é mínimo e cada serviço pode ser escrito usando uma linguagem de programação/tecnologias/bancos de dados distintos.

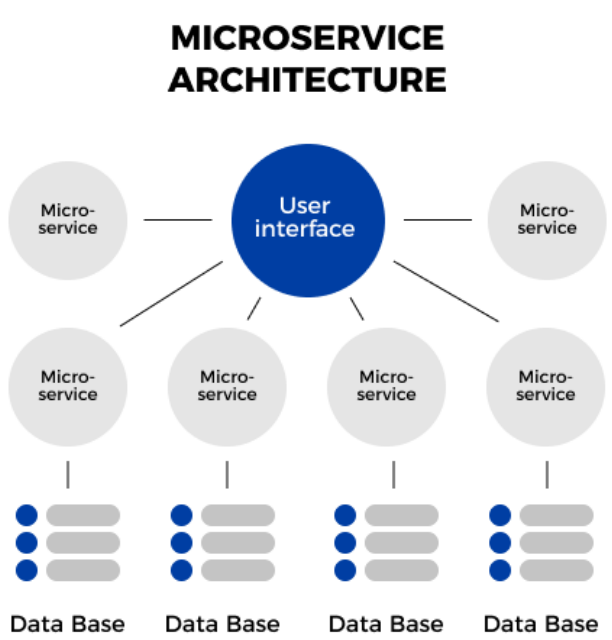
Princípio

- Micros serviços baseiam-se no princípio da componentização de serviços
- Decompõem o software em vários componentes que podem ser definidos como serviços
  - Responsabilidade única
  - Isolado por natureza
  - Alterações em um serviço não deve afetar os demais.

Componentes

- Serviços
- “Service bus”
- Configuração externa
- API Gateway
- Containeres

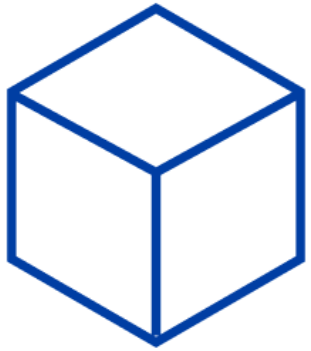




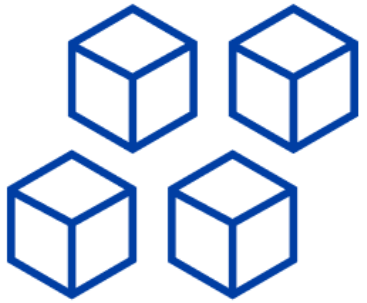
Características

- Componentização através de serviços
- Organizada ao redor de capacidades de negócio
- Produtos, não projetos
- “Endpoints” inteligentes
- Governança descentralizada
- Gestão de dados descentralizada
- Automação da infraestrutura
- Projeto visando falhas
- Projeto evolucionário

Comparação de modelos



MONOLITHIC
Single unit



SOA
Coarse-grained

MICROSERVICES
Fine-grained

Vantagens e desvantagens

- Baixo acoplamento
- Modularidade
- Falha em um serviço não impacta os demais
- Alta flexibilidade
- Alto grau de escalabilidade
- Facilidade de modificação

- Grande chance de falha na comunicação entre os serviços
- Dificuldade de gerenciar grande número de serviços
- Necessidade de resolver problemas tais como latência de rede, balanceamento de carga e outros problemas de arquiteturas distribuídas
- Teste mais complexo sobre um ambiente distribuído
- Implementação requer mais tempo

Conclusão

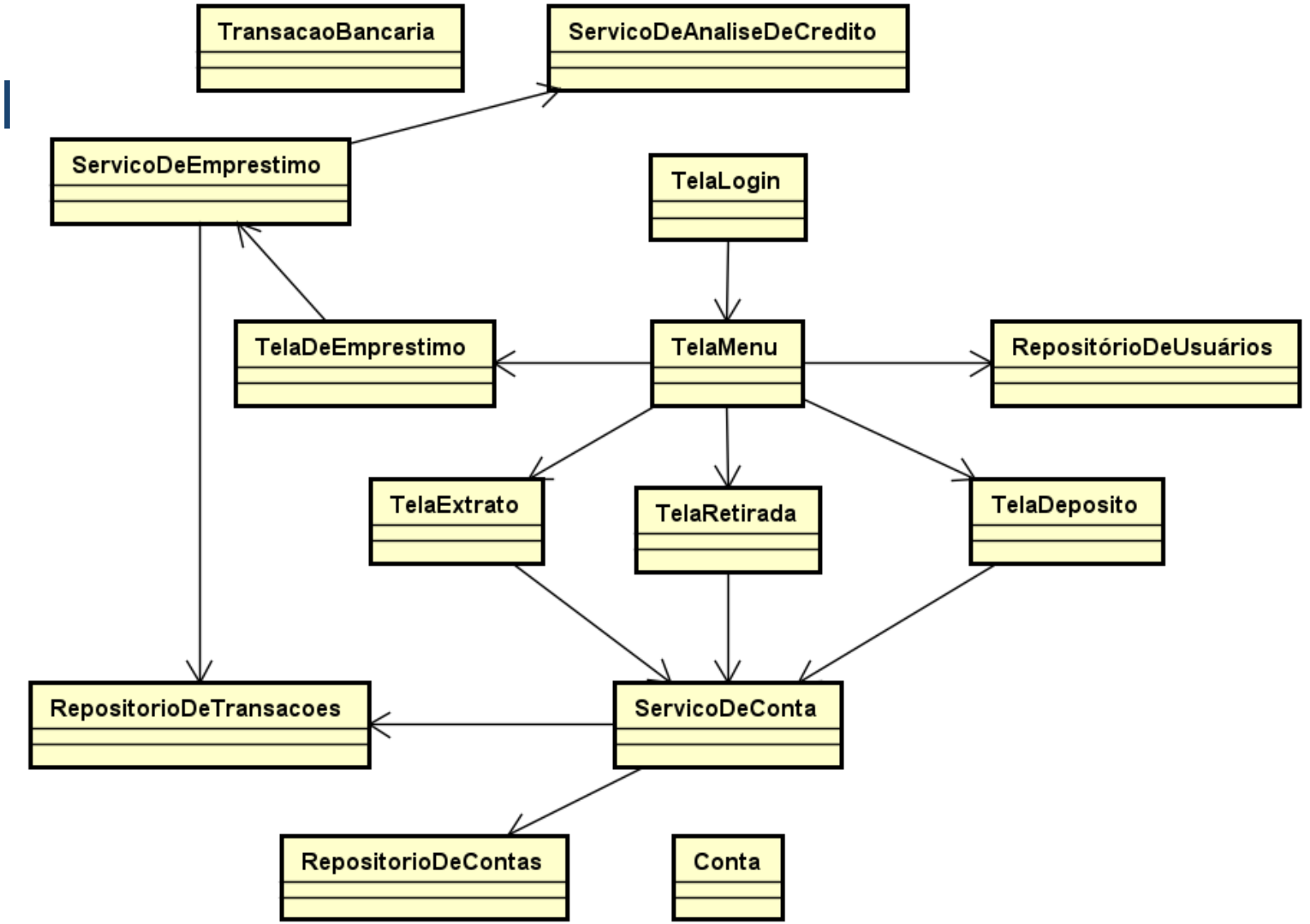
- Though there is no perfect software architecture in existence, but any architectural approach can be considered relatively perfect if it fulfils the functional and nonfunctional needs of the Project”

Dinâmica

D1) Com base na descrição do sistema bancário que segue, analise o conjunto de classes apresentado no próximo slide e organize as classes em camadas coesas segundo o padrão “camadas”. Acrescente classes se julgar necessário.

Especificação do sistema: em um sistema bancário os usuários tem acesso ao menu principal após fazerem login. Neste menu existem opções para acessar a tela de depósitos, a tela de retiradas, a tela de extrato e a tela de empréstimos. Nas telas de depósito ou retirada basta informar o valor da operação que a transação é registrada e o saldo atualizado (no caso da retirada verifica-se antes se o saldo é suficiente para a operação). Uma transação armazena a data, o tipo de operação, o número da conta e o valor. A tela de extrato mostra automaticamente o extrato dos últimos 30 dias. Já a tela de empréstimo solicita o valor desejado e o prazo para pagamento e faz análise de crédito baseado no saldo médio do cliente no último ano (se o cliente tiver menos de um ano de transações registradas o empréstimo não é concedido).

MARISTA


TransacaoBancaria
ServicoDeAnaliseDeCredito
ServicoDeEmprestimo
TelaLogin
TelaDeEmprestimo
TelaMenu
RepositórioDeUsuários
TelaExtrato
TelaRetirada
TelaDeposito
RepositorioDeTransacoes
ServicoDeConta
RepositorioDeContas
Conta