Visão Geral Sobre Arquitetura de Software

Panorama geral dos problemas e aspectos envolvidos

Arquitetura de Software

- Para criar um produto confiável, seguro e eficiente, deve-se prestar atenção aos seguintes tópicos:
  - Sua organização geral
  - Como o software é decomposto em componentes
  - A organização dos servidores
  - A tecnologia que você usa para construir o software.
- Em resumo, deve-se projetar a **Arquitetura do Software**

Relembrando

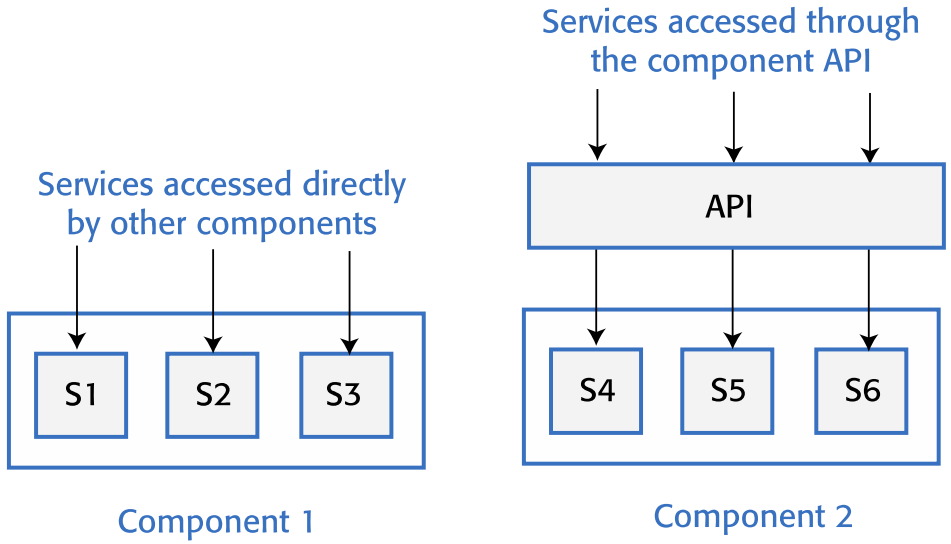
- Arquitetura é a organização fundamental de um sistema de software embutida em seus **componentes**, seus relacionamentos entre si e com o ambiente e os princípios que guiam seu design e evolução

Arquitetura de software x componentes

- “Componente” é um termo muito geral: pode ser tanto um programa (grande escala) como um objeto (pequena escala).
- Um componente é um elemento que implementa um conjunto coerente de funcionalidades ou características
- Quando se projeta uma arquitetura de software, não é necessário decidir como um elemento arquitetural ou componente será implementado
- Ao invés disso, projeta-se a interface do componente e deixa-se a implementação daquela interface para um segundo momento dentro do processo de desenvolvimento

Componentes como uma coleção de serviços

- Um componente pode ser entendido como uma coleção de um ou mais serviços que podem ser usados por outros componentes
  - Componente de grande escala: um SGBD
  - Componente de pequena escala: um micro serviço que faz algo muito específico.
- Podem ser objetos acessados por outros objetos ou podem ser serviços acessados através de uma API.



Porque arquitetura é importante?

- Porque influencia as propriedades não funcionais do sistema
- Quando se desenvolve um protótipo, as propriedades não funcionais não são importantes
- Quando se desenvolve um produto as propriedades não funcionais se tornam essenciais

MARISTA

Atributos de qualidade não funcionais

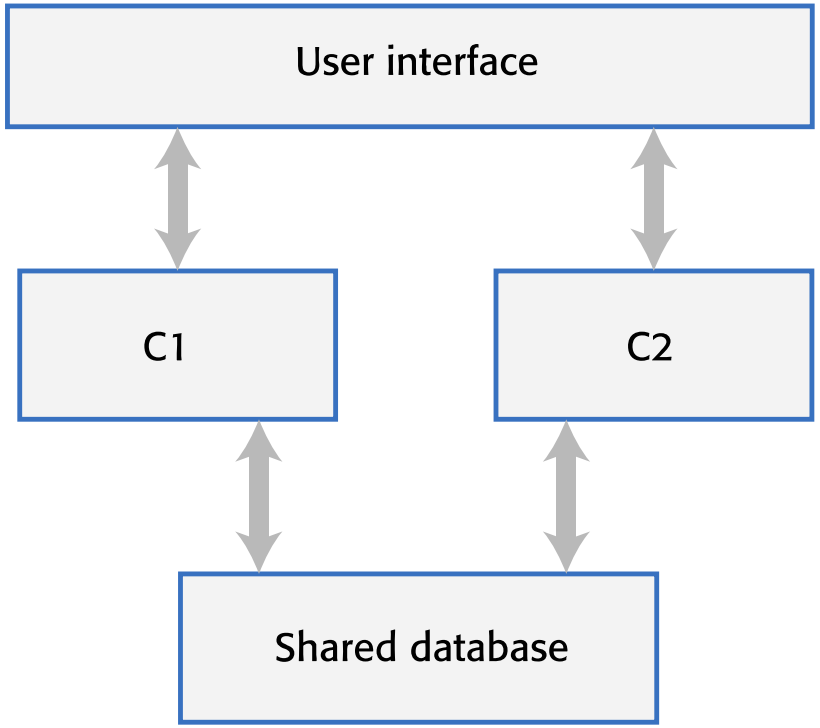
- Responsividade
  - O sistema responde em um tempo razoável?
- Confiabilidade
  - As funcionalidades se comportam como esperado?
- Disponibilidade
  - O sistema consegue entregar seus serviços quando solicitado?
- Usabilidade
  - Os usuários conseguem acessar os recursos desejados rapidamente e sem erros?
- Manutebilidade
  - O sistema pode ser atualizado rapidamente e novas funcionalidades adicionadas sem um custo excessivo?
- Resiliência
  - O sistema consegue continua entregando seus serviços mesmo após uma falha parcial ou ataque externo?

Exemplo 1: influência dos requisitos não funcionais

- Arquitetura de segurança centralizada: Star Wars → Rogue One
- Os planos do equipamento do "império" foram armazenados em um único local altamente seguro
- Bastou conseguir quebrar essa segurança para os rebeldes terem acesso (indevido) aos dados
- Se os planos estivessem armazenados de forma distribuída:
  - Seria mais difícil consulta-los quando necessário
  - Seria muito mais complicado fazer um acesso indevido porque implicaria quebrar a segurança de vários locais

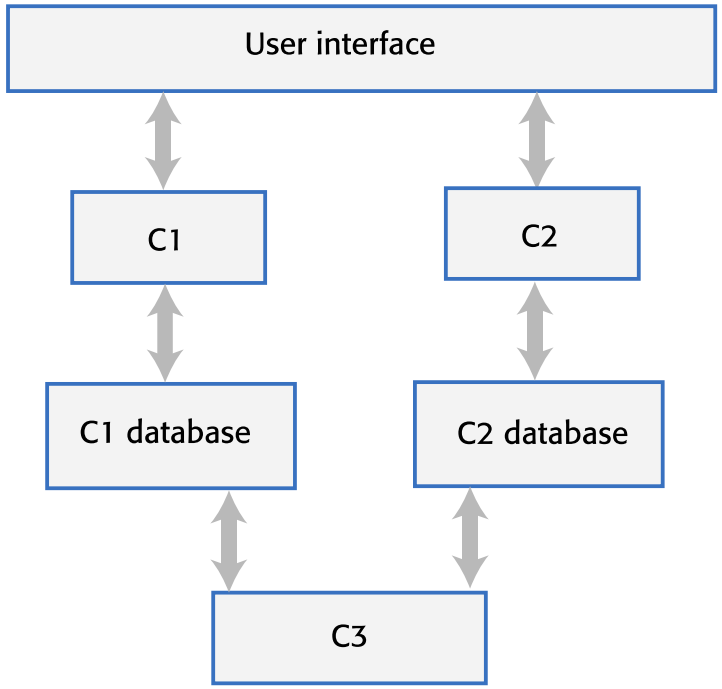
Exemplo 2: influência dos requisitos não funcionais

- Problema:
  - Os componentes C1 e C2 compartilham o BD
  - Assumindo que C1 roda lentamente porque reorganiza a informação no BD antes de usar ela.
  - A única forma de C1 ser mais rápido é alterar a organização do BD. Isso significa que C2 também tem de ser alterado, o que pode afetar seu tempo de resposta



Exemplo 2: influência dos requisitos não funcionais

- Arquitetura alternativa (solução):
  - Cada um dos componentes tem cópia das partes do banco de que necessitam
  - Se um componente tem de alterar a organização do banco, isso não afeta o outro componente
  - O componente C3 passa a ser necessário para garantir que os dados compartilhados por C1 e C2 sejam mantidos consistentes
- Entretanto uma arquitetura com vários BDs pode ser mais cara de implementar → decisão de projeto arquitetural



Database reconciliation

Figura: Ian Sommerville/2019

Projeto Arquitetural

- Não é possível otimizar todos os aspectos não funcionais no mesmo sistema.
- A otimização de um atributo tal como segurança envolve afetar outros tais como usabilidade ou eficiência
- O balanço entre essas questões é um dos aspectos fundamentais do projeto arquitetural
- Outra questão fundamental é minimizar a complexidade
  - Quanto mais complicado mais difícil de alterar e manter
  - Quanto mais complicado mais fácil de inserir erros

MARISTA

Projeto arquitetural = escolhas

- Vários aspectos humanos e organizacionais impactam a arquitetura de um software: cronograma de lançamento, capacidade da equipe, orçamento.
- O projeto arquitetural envolve “balancear” estas questões buscando a melhor arquitetura dentro das condições que se apresentam.
- Exemplos:
  - Manutebilidade x performance
  - Segurança x usabilidade
  - Disponibilidade x “time to Market”

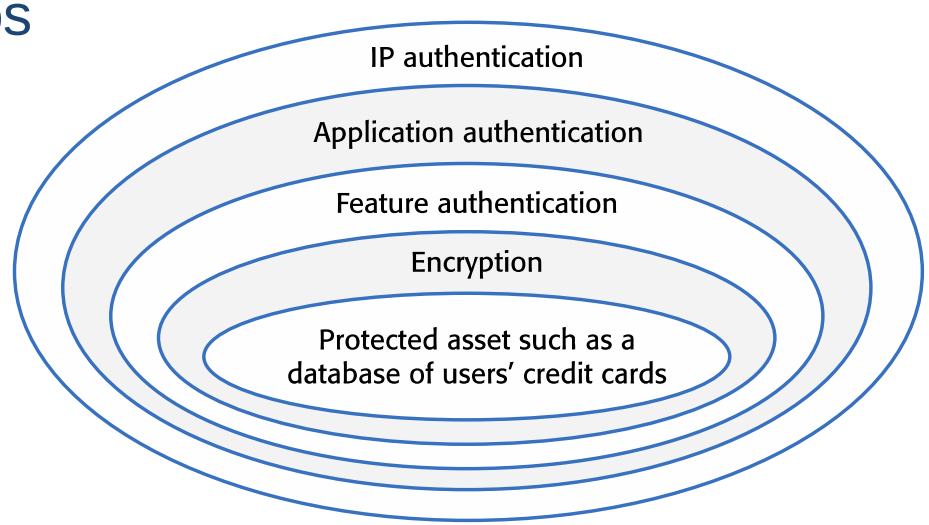
Problema: Manutebilidade x Performance

- Manutebilidade é um atributo que reflete o quão difícil e caro é fazer alterações no sistema depois que ele foi entregue para o cliente
- Pode-se melhorar a Manutebilidade construindo o sistema a partir de partes autocontidas
- Em termos arquiteturais isso significa que o sistema deve ser decomposto em componentes de grão fino, cada um fazendo uma única coisa e uma coisa apenas
- Entretanto isso exige tempo para que os componentes se comuniquem entre si o que podem implicar que o software fique mais lento.

MARISTA

Problema: segurança x usabilidade

- Pode-se obter segurança projetando um sistema de proteção com uma série de camadas
- Camadas podem ter níveis de autenticação, encriptação e assim por diante
- Do ponto de vista arquitetural eles podem ser implementados como componentes separados de maneira que se um for violado os demais permanecem intactos



MARISTA

Problema: segurança x usabilidade

- A abordagem de segurança por níveis pode afetar a usabilidade do software
- O usuário pode ter de guardar informações (senhas) necessárias para entrar em um determinado nível. A interação fica mais lenta devido aos recursos de segurança
- Os usuários costumam se irritar com este tipo de segurança excessiva
- Para evitar isso será necessária uma arquitetura:
  - Que não tenha muitos níveis de segurança
  - Que não force segurança desnecessária
  - Que forneça componentes que reduzam a carga sobre o usuário

MARISTA

Problema: disponibilidade x time-to-market

- Disponibilidade é especialmente importante em sistemas 24x7
- Corresponde ao tempo em que permanece “no ar”
- Normalmente é medida em porcentagem do tempo em que os serviços estão disponíveis
- Do ponto de vista arquitetural se obtém disponibilidade dispondo de componentes redundantes
- Implementar componentes redundantes requer tempo e aumenta o custo de desenvolvimento. Acrescenta complexidade no sistema e aumenta a chance de serem inseridos erros e vulnerabilidades

MARISTA

Projeto arquitetural: 3 questões

- Uma vez decididos quais os atributos não funcionais mais importantes, o projeto arquitetural terá de lidar com 3 questões:
  - De que forma o sistema será decomposto em componentes funcionais
  - Como os componentes serão distribuídos e como irão se comunicar entre si
  - Que tecnologias serão usadas na construção do sistema

MARISTA

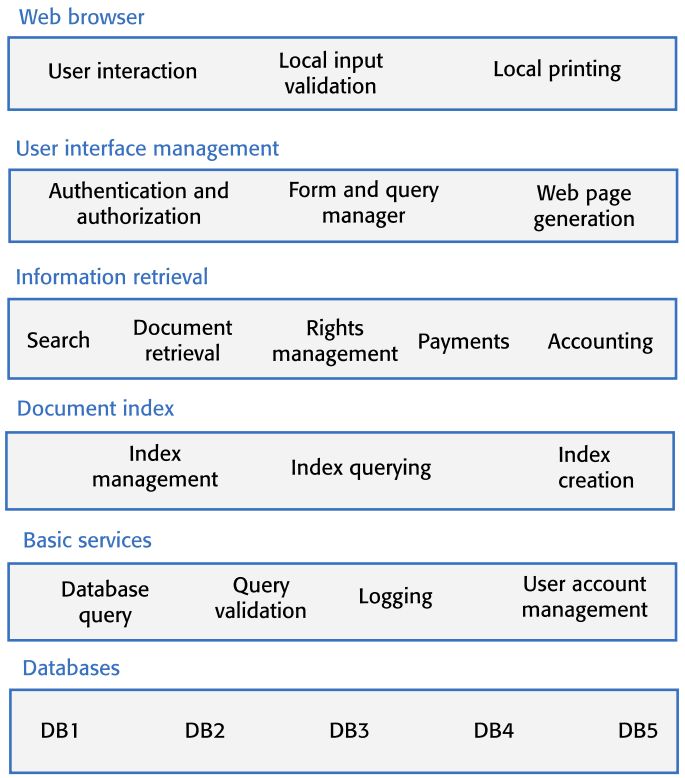
Questão 1: decomposição

- Abstração é fundamental em projeto de software: concentrar-se nos aspectos essenciais sem se preocupar com detalhes
- Em um primeiro momento a decomposição deve ser em componentes de larga escala
- Decomposição envolve analisar os componentes de larga escala e então representar eles como um conjunto de componentes mais refinados

MARISTA

Exemplo:

- Sistema de recuperação de documentos armazenados em bancos de dados privados
- O sistema deveria controlar os direitos de acesso e cobrar os acessos pagos
- Um modelo informal em camadas foi usado para mostrar como o sistema foi decomposto em componentes funcionais.
- Em sistemas Web o fluxo parte das ações do usuário (eventos) então o fluxo é de cima para baixo.
  - A partir de um evento, a informação é criada nos níveis inferiores, transformada nos níveis intermediários e entregue para os usuários no nível mais alto



MARISTA

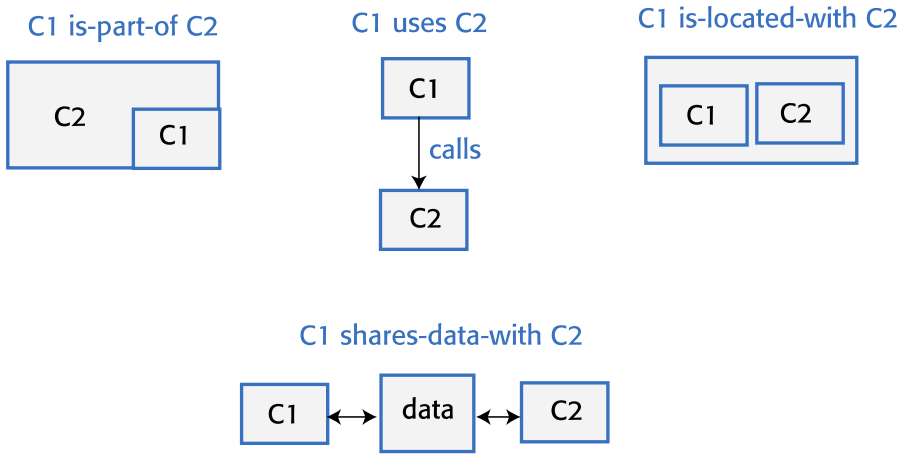
Decomposição: taxonomia

- Serviço: uma unidade coerente de funcionalidade. Pode significar coisas diferentes em níveis diferentes do sistema. Exemplo um sistema pode oferecer um “serviço de email”. Este pode incluir, por sua vez, serviços de criação, envio, recebimento e armazenamento de emails
- Componente: é o nome de uma unidade de software que provê um ou mais serviços. Quando usado por outros serviços é acessado através de uma API. Pode usar outros componentes para implementar seu propósito.
- Módulo: um agrupamento de componentes relacionados. Por exemplo, podem prover um conjunto de serviços relacionados.

MARISTA

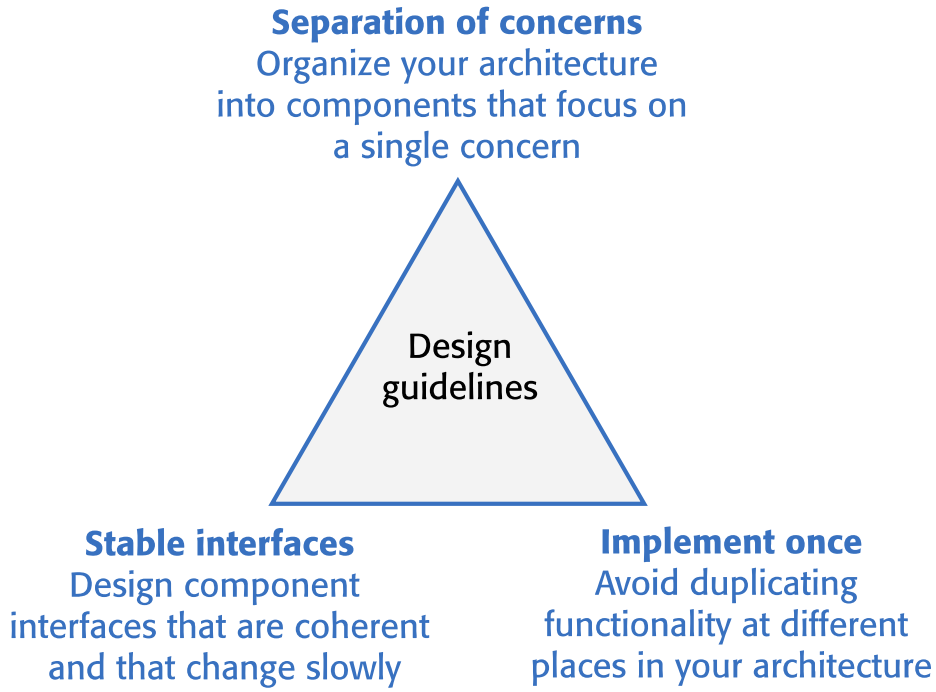
Decomposição: complexidade

- A complexidade de uma arquitetura vem da quantidade e da natureza dos relacionamentos entre os componentes
- Quando se decompõem um sistema deve-se evitar a complexidade desnecessária
  - Localize no mesmo módulo componentes que tem relacionamentos entre si
  - Prefira usar dados locais e evite compartilhar dados se possível



Decomposição: orientações gerais

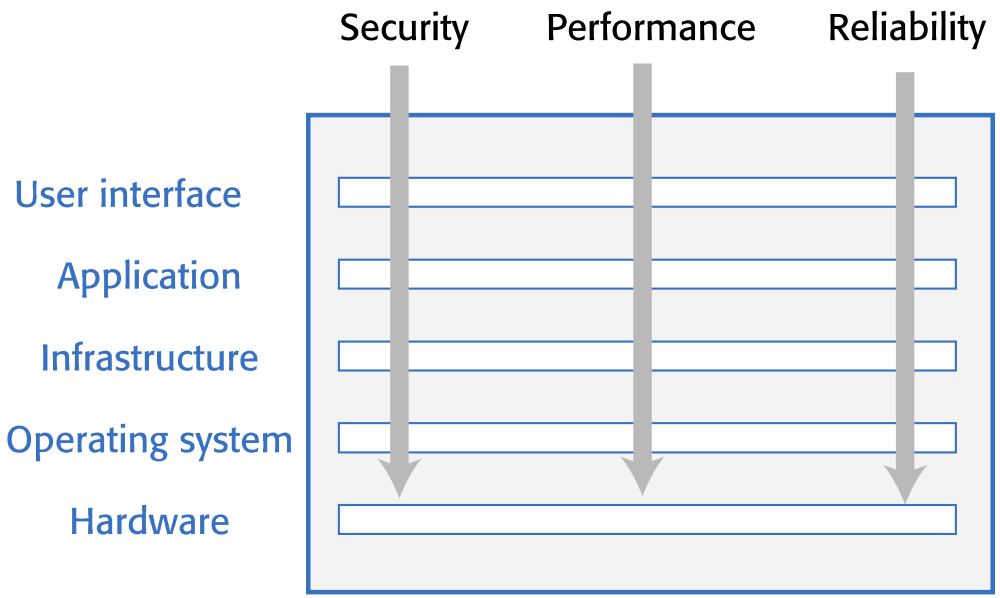
- Separar as responsabilidades
- Evitar duplicação de funcionalidades
- Manter as interfaces estáveis



MARISTA

Decomposição em camadas

- As camadas não são componentes ou módulos e sim agrupamentos lógicos de componentes
- Todas as dependências devem ser das camadas superiores para as inferiores
- Sempre que possível cada responsabilidade deve estar concentrada em uma única camada
- Alguns aspectos, entretanto, são transversais
  - Funcionalidades em geral são melhor ajustadas nas camadas
  - Propriedades não funcionais costumam ser transversais



MARISTA

Decomposição de sistemas WEB

- Arquiteturas WEB são basicamente arquiteturas cliente-servidor
- Modelo tradicional:
  - Browser → HTTP → servidor retorna uma página pronta para ser exibida pelo browser

Organização em camadas

- Camadas lógicas
  - Separação de competências
  - Facilidade de manutenção
- Camadas físicas
  - Questão de implantação
  - Performance, escalabilidade, disponibilidade, manutenção

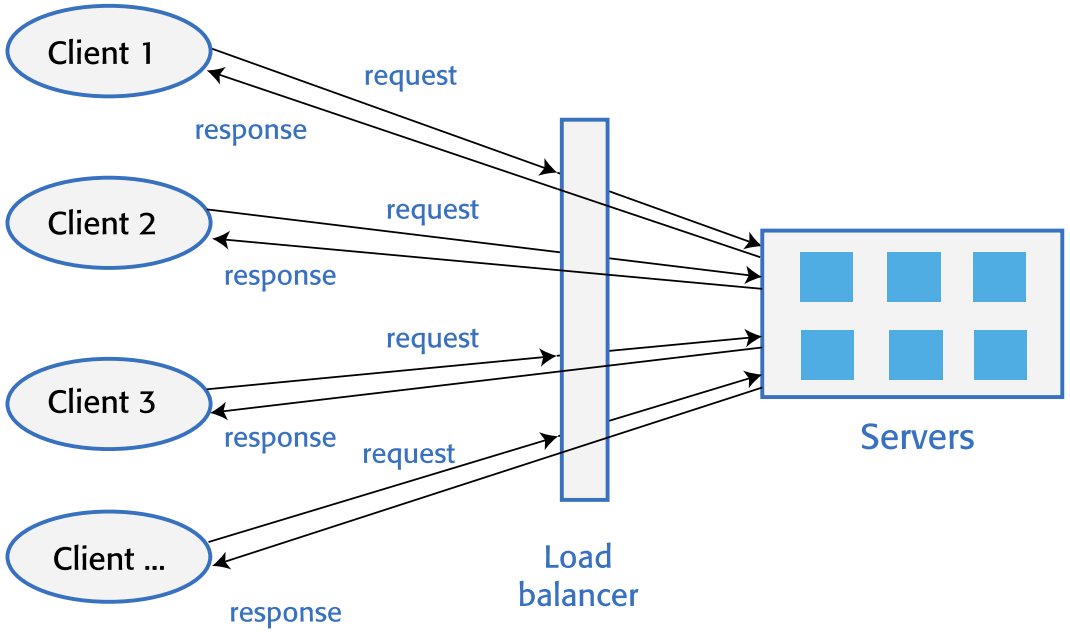
Questão 2: distribuição

- A maioria dos sistemas desenvolvidos atualmente são sistemas WEB usando uma arquitetura cliente-servidor
- Neste tipo de arquitetura:
  - A interface com o usuário é implementada no próprio dispositivo cliente (computador, mobile, gadget)
  - As funcionalidades são distribuídas entre o cliente e um ou mais servidores
- A arquitetura do sistema deve definir a quantidade de servidores e a alocação dos componentes nesses servidores

MARISTA

Distribuição: cliente servidor

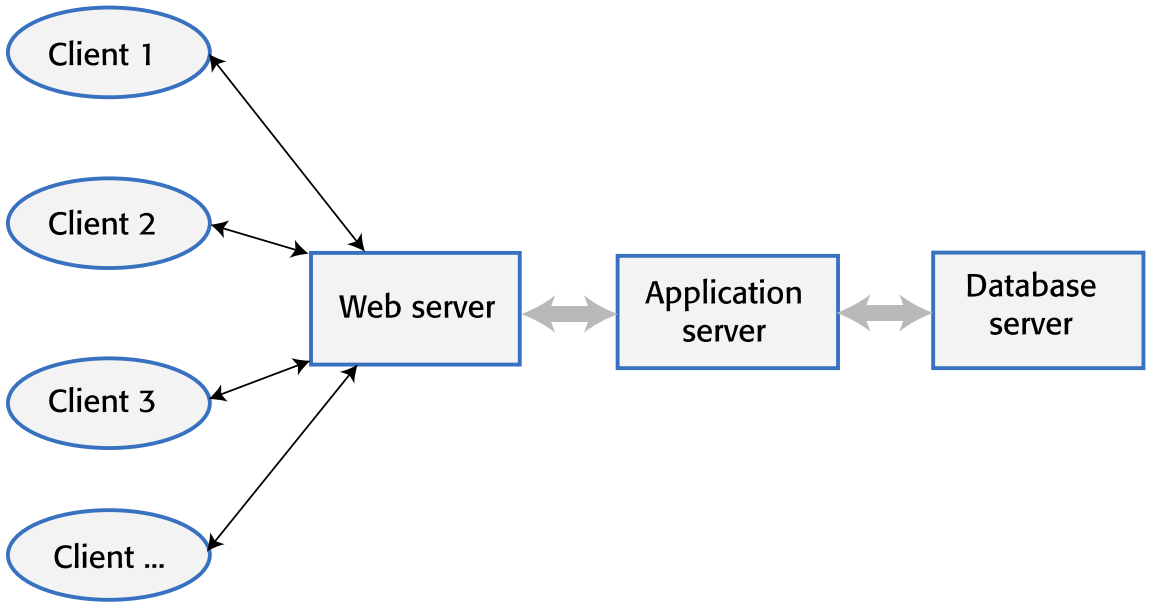
- Arquiteturas cliente servidor são adequadas para aplicações onde clientes compartilham um banco de dados e operações lógicas sobre estes dados
- Este tipo de arquitetura pode ter um ou vários servidores
- O acesso aos servidores é feito através de um balanceador de carga



MARISTA

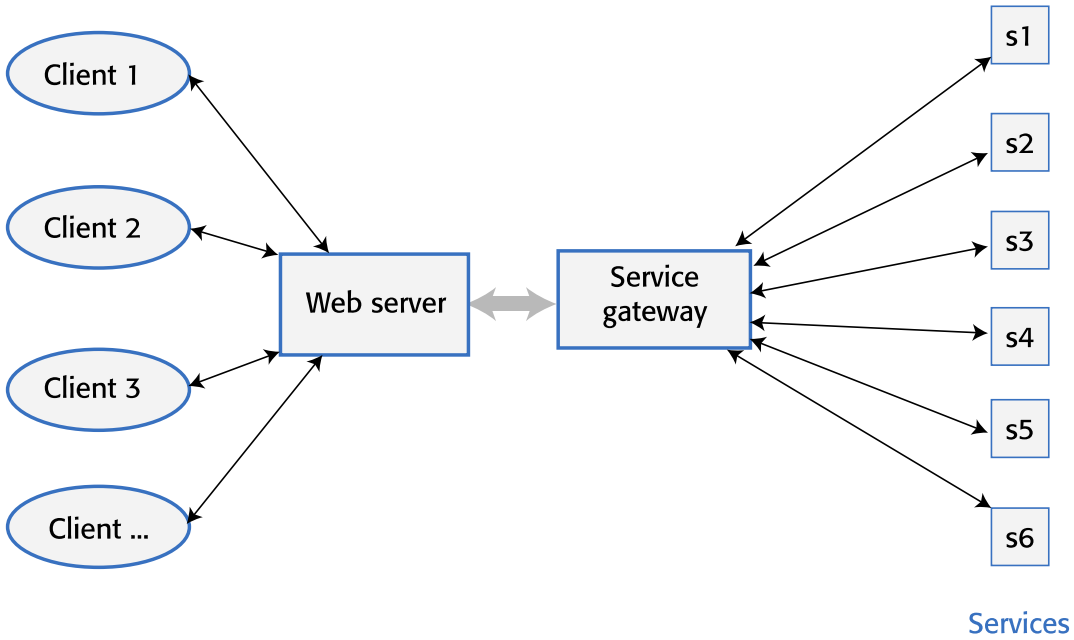
Distribuição: multicamadas físicas

- Arquiteturas com várias camadas físicas usam servidores especializados adicionais



Distribuição: micros serviços

- Serviços, neste tipo de arquitetura, são componentes sem estado, o que significa que podem ser replicados e migrar de um computador para outro
- Muitos servidores podem estar envolvidos em providenciar serviços
- Uma arquitetura orientada a serviços normalmente é mais fácil de escalar a medida que a demanda cresce e é mais resistente a falhas



Questões envolvidas na seleção da forma de distribuição

- Tipos de dados e atualização dos dados
  - Se estão sendo usados dados estruturados atualizados por diferentes funcionalidades, normalmente é mais simples ter um único banco de dados compartilhado que forneça mecanismos de bloqueio e gerenciamento de transações. Se os dados são distribuídos entre serviços será necessário dispor de uma maneira de manter eles consistentes o que adiciona “overhead” ao sistema
- Alterações frequentes
  - Se for possível antecipar que os componentes serão trocados com frequência, então isolar estes componentes simplifica a manutenção
- Plataforma de execução do sistema
  - Se o plano é que o sistema execute na nuvem, então o melhor é usar uma plataforma orientada a serviços
  - Se o sistema executa em servidores locais, então uma arquitetura com várias camadas físicas (multi-tier) pode ser mais apropriada

Tecnologia: questões

- Banco de dados
  - SQL: gerenciamento de transações; estruturas simples e previsíveis; consistência das informações
  - NoSQL: informações desestruturadas; muitas operações de leitura e análise de dados; pouca atualização
- Plataforma de entrega
  - Em tablets e smartfones ao contrário de computadores é preciso considerar: conectividade intermitente, capacidade do processador, gestão de energia e teclado na tela.
- Servidores
  - Servidores individuais: questões de segurança
  - Servidores na nuvem: implica em desenhar uma arquitetura orientada a serviços e usar a API do fornecedor → escalabilidade e resiliência (capacidade de recuperação)