

# CONTEÚDO DA AULA



Nessa aula você aprenderá:

Modelos de Arquitetura Desenhos de Arquitetura Ciclo de Vida de Produto



#### Palavra-chave dessa Sprint:

## **PRAGMATISMO**

#### prag·má·ti·co

- . adjetivo
- 1. Relativo à pragmática ou ao pragmatismo.
- 2. Que tem motivações relacionadas com a ação ou com a eficiência. = PRÁTICO
- . adjetivo e substantivo masculino
- 3. Que ou quem revela um sentido prático e sabe ou quer agir com eficácia.





## Frase dessa sprint:

Aprender/Ensinar processos, métodos e ferramentas para construção e manutenção de softwares profissionais.





#### C4 Model

É uma abordagem de modelagem de Arquitetura de Software que fornece uma estrutura visual composta por 4 níveis de abstração:

- Contexto: Visão mais alta, descreve o sistema e o ambiente.
- Contêineres: Descreve os principais contêineres do sistema (banco de dados, back-end, front-end, APIs)
- Componentes: Descreve os componentes dentro dos contêineres.
- Código: Visão mais detalhada, apresenta do diagrama de classes.

### Conceitos que serão utilizados

Vamos pensar em containers (<u>não é Docker</u>), mas pensar que o **container é conjunto que precisa estar funcionando ou rodando para um software funcionar**.

Exemplos de Containers (Representados por grandes quadrados):

Server-side web application: Aplicação backend. Ex: Spring MVC, NodeJs, Asp.NET MVC, etc.

**Client-side web application**: A aplicação Javascript que roda no Web Browser. Ex: Angular, JQuery,

React.

Client-side desktop application: A aplicação que roda local. Ex: Java JAR, .NET Windows, C++.

**Mobile app**: Ex: App IOS, App Android, App React Native.

**Server-side console application**: Ex: "public static void main" application, batch, script.

**Microservice**: Ex: Spring Boot.

**Serverless function**: Uma função que independe se servidor. Ex: Amazon Lambda, Azure Function.

Database: Um banco de dados relacional ou de objetos. Ex: MySQL, SQL Server, Oracle Database,

MongoDB.C

## **Atividade Lego**

#### **API Externa**

[Container: XXXX]

Facilidade ?
Aprendizado:?
Flexibilidade:?
Confiabilidade:?
Tempo de Mercado:?

#### **Database**

[Container: XXXX]

Facilidade Aprendizado: Flexibilidade: Confiabilidade Tempo de Mercado

#### Back-end

[Container: XXXX]

Facilidade
Aprendizado:
Flexibilidade:
Confiabilidade
Tempo de Mercado

#### Back-end

[Container: XXXX]

Facilidade
Aprendizado:
Flexibilidade:
Confiabilidade
Tempo de Mercado

#### **API Externa**

[Container: XXXX]

Facilidade ?
Aprendizado:?
Flexibilidade:?
Confiabilidade:?
Tempo de Mercado:?

#### Front-end

[Container: XXXX]

Facilidade Aprendizado:

Flexibilidade: Confiabilidade Tempo de Mercado

#### **API Externa**

**API Externa** 

[Container: XXXX]

Facilidade?

Aprendizado:?

Flexibilidade:?

Confiabilidade:?

Tempo de Mercado:?

[Container: XXXX]

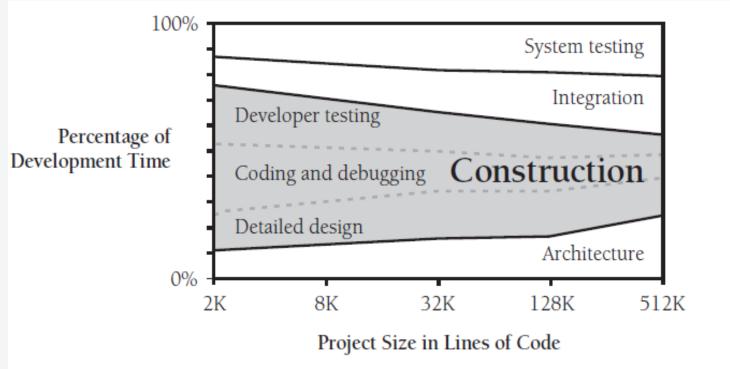
Facilidade ?
Aprendizado:?
Flexibilidade:?
Confiabilidade:?
Tempo de Mercado:?

# O Desenho de Arquitetura

de Software é essencial

em grandes projetos!

### Por que a Eng. de Software é importante?



Livro Code Complete pag 654

A medida que o tamanho do sistema aumenta, o tempo nas atividades de Arquitetura, Integração e Teste também aumentam.

O tempo da construção do software reduz proporcionalmente

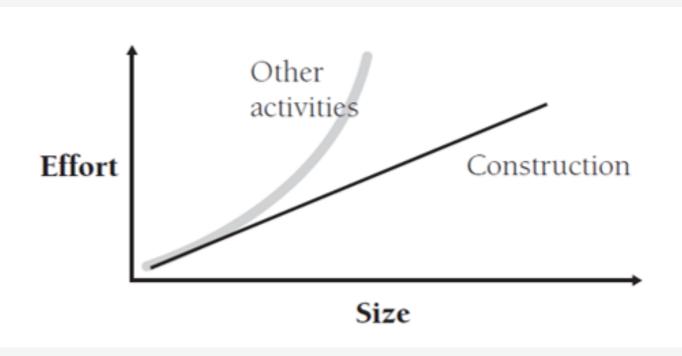
#### **Atividades que aumentam:**

- Comunicação
- Planejamento
- Gerenciamento
- Lev. Requisitos

- Projeto Funcional
- Arquitetura
- Integração
- Remoção de Defeitos

- Testes
- Documentação

#### Por que a Eng. de Software é importante?



Quando você faz um castelo a chance de esquecer a porta aumenta muito!





## Por que a Eng. de Software é importante?



# Existem DIVERSOS modelos de Arquitetura

Vamos conhecer alguns...

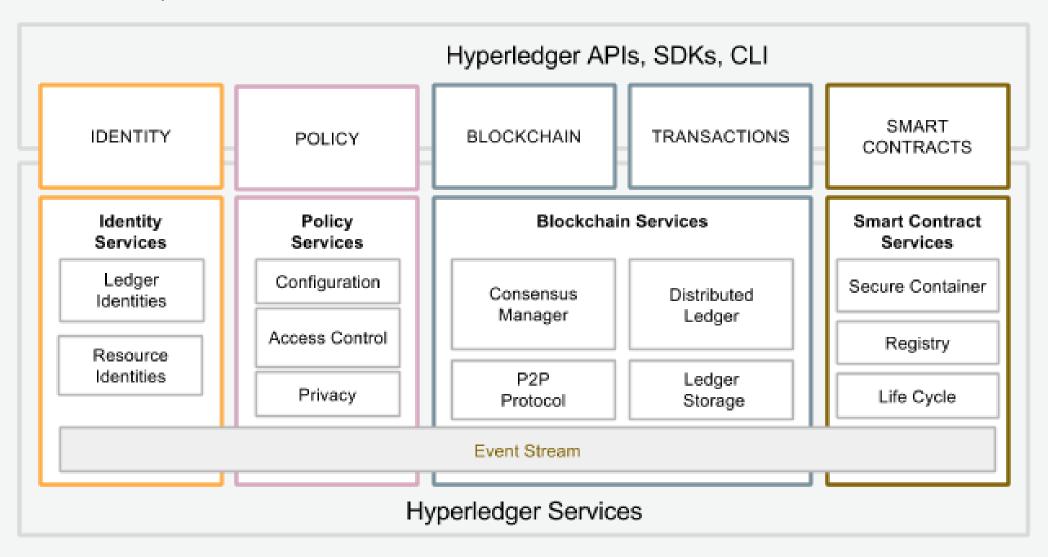
### O Desenho não precisa ser complexo...

Desenho de arquitetura de um sistema....

Camada de Sistema - Bases de Dados Camada de Sistema - Integrações Camada de Sistema - Aplicação Aplicação Mobile Aplicação WEB

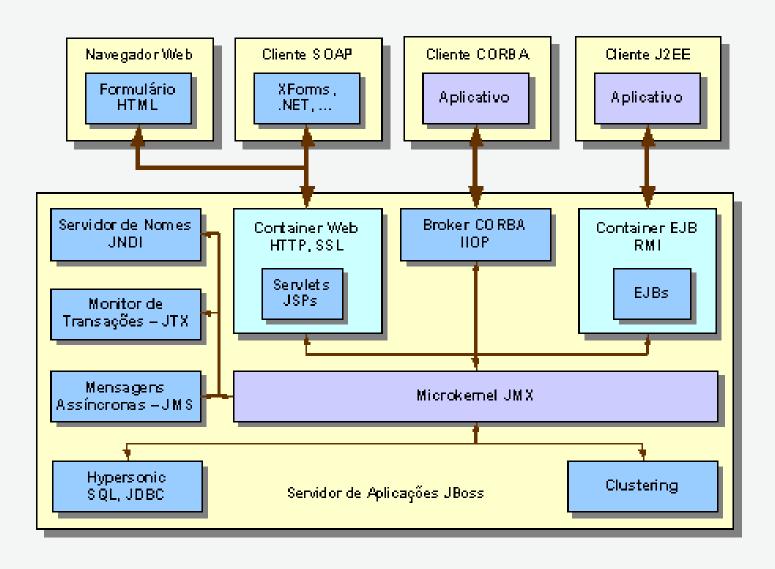
#### Mas muitos podem precisar ser...

Desenho de arquitetura de um sistema de Block Chain....



## Mas muitos podem precisar ser...

Desenho de arquitetura do JBoss....



### Perguntas para decidir arquitetura [Sommerville]

- 1. Existe uma arquitetura conhecida que pode funcionar com o projeto?
- 2. Como o sistema vai utilizar as diversas capacidades de processamento (disponibilidade)?
- 3. Qual o estilo mais apropriado para a solução (cliente-servidor, 2 camadas, 3 camadas, etc).
- 4. Como será a decomposição do sistema? (módulos)
- 5. Como será o controle da operação do sistema? Centralizado, baseado em Eventos?
- 6. Como o projeto de arquitetura será avaliado?
- 7. Como vai ser a documentação?

...essas são questões fundamentais, outras coisas podem impactar como: mão de obra, maturidade da solução....

## Arquitetura pode tratar de ...

PROTOCOLOS DE COMUNICAÇÃO

**EVOLUÇÃO DO SISTEMA** 

# INTERAÇÃO ENTRE OS COMPONENTES

**PRODUTOS** 

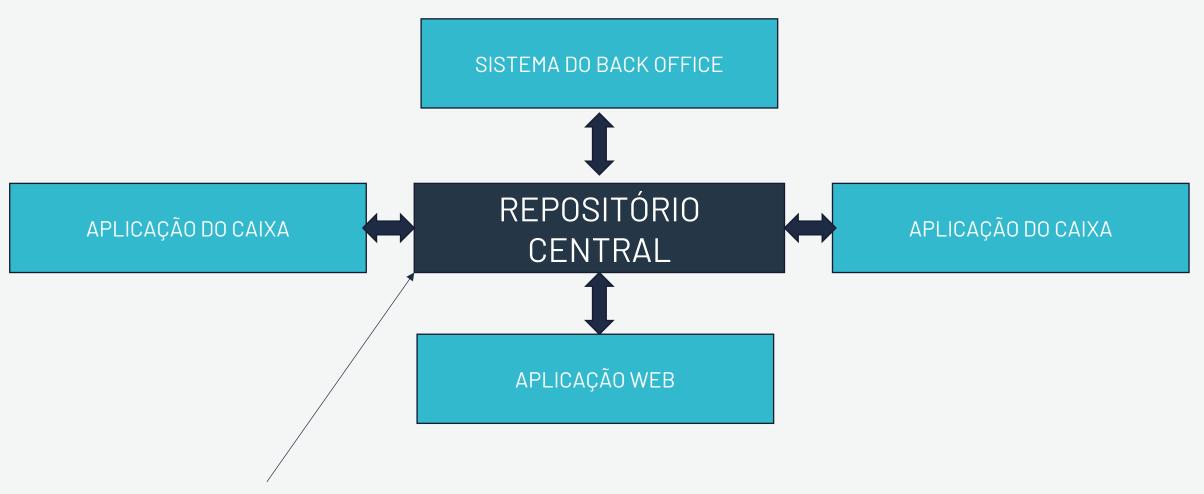
HARDWARE

CONECTIVIDADE

**FABRICANTES** 



## Modelos de Repositório



Tudo depende dele, se ele parar, tudo para. (exemplo: DB, se cair...)

## Modelos de Repositório

Eficiente para compartilhar grandes quantidades de dados

Os sistemas precisam estar de acordo com o modelo de dados

Não há necessidade do sistemas saberem como os demais utilizam os dados

Evoluir o modelo pode ser difícil

Atividades de backup, controle de acesso e recuperação de erros são centralizadas

As políticas (ex: backup) são impostas para todos os sistemas consumidores

Um sistema novo ou ferramenta nova podem ser integradas diretamente (usando o modelo de dados)

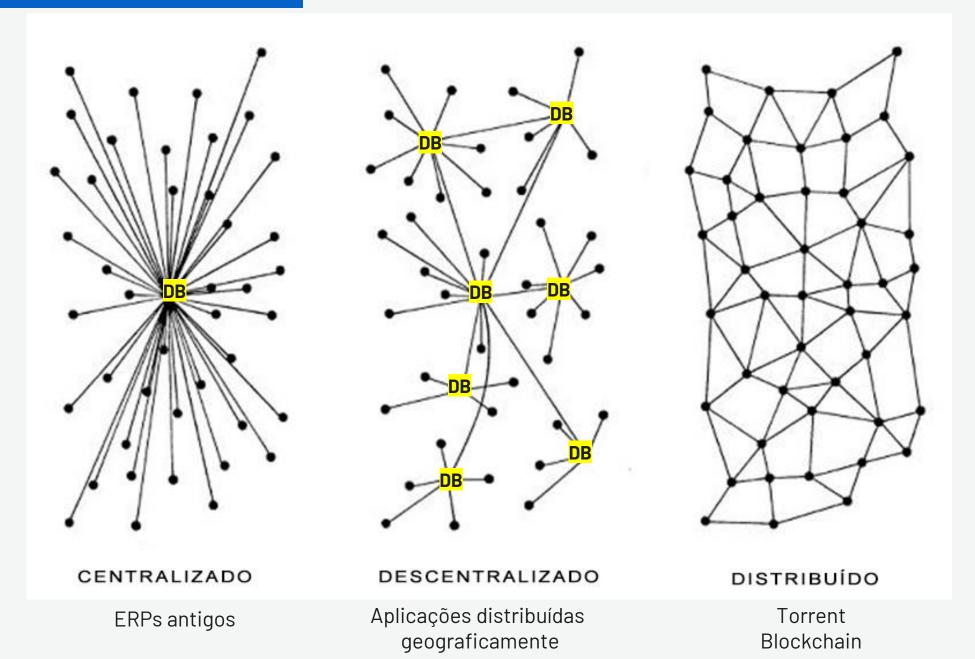
Pode ser difícil distribuir os dados para diversas máquinas (ex: geograficamente)

ATENÇÃO: Se precisa de manutenção no Repositório Central, para todo mundo!

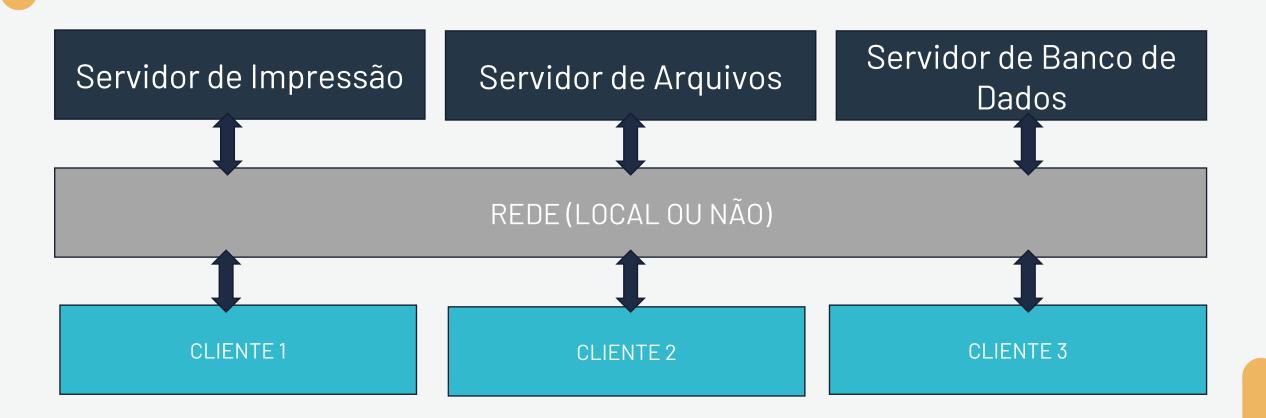


Muitos sistemas corporativos usam esse modelo

## Tendências....



### Modelo Cliente Servidor



#### **Modelo Cliente Servidor**

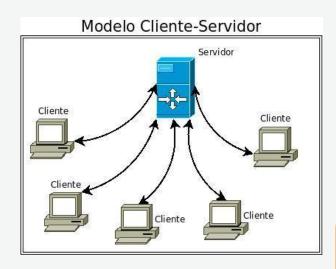
Um conjunto de servidores fornecem serviços para outros sistemas

Um conjunto de clientes solicitam serviços para os servidores

Existe uma rede para viabilizar o acesso

O cliente e o servidor podem estar na mesma máquina, mas isso não é prática





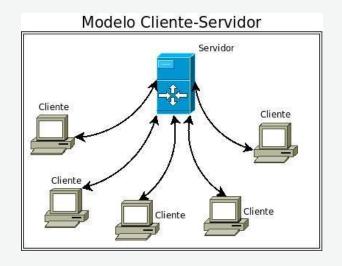
#### Benefícios: Modelo Cliente Servidor

Segurança dos dados\*

Acesso centralizado aos dados

Fácil manutenção dos dados





#### Modelo em Camadas

> Juntar as coisas parecidas, que tem responsabilidades parecidas em uma mesma camada.

Camada de Sistema – Gerenciamento dos dados Tudo que vai <u>controlar os dados, fica na camada de</u> <u>gerenciamento de dados.</u>

Camada de Sistema – Processamento da Aplicação Tudo que está relacionado com <u>regras de negócio, fica na</u> <u>camada de sistemas</u>.

Camada de Sistema -Apresentação Tudo que está relacionado com apresentação, por exemplo, o <u>Front-end, fica na camada de apresentação</u>.

#### Modelo em Camadas - Muita coisa é feita em camada

Cada camada pode ser imaginada como um servidor/máquina abstrata

Camada OSI de redes é um exemplo de camada



O modelo de camadas apoia o modelo de componentes e desenvolvimento incremental dos sistemas

Como desvantagem, a estruturação do sistema pode ser mais difícil, assim como o gerenciamento da configuração

O desempenho também deve ser considerado quando utilizado um número grande de camadas (atenção especial para a rede).



## Benefícios: Modelo em Camadas

Deploy facilitado

Redução de custos

Camada de Sistema – Bases de Dados

Camada de Sistema - Integrações

Camada de Sistema - Aplicação

Aplicação Mobile

Aplicação WEB

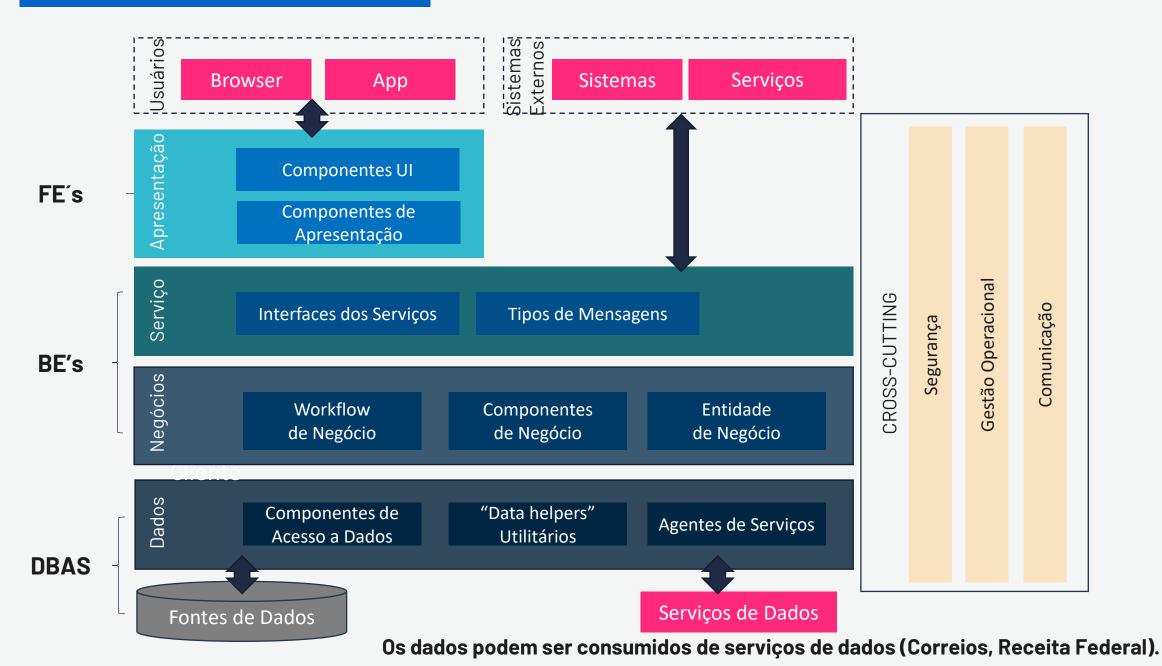
Desenvolvimento facilitado

Reutilização

Mitigação da complexidade técnica (Separação dos conceitos)



#### Modelo em Camadas



#### Cliente Gordo vs Cliente Magro

Regras de negócio embarcadas Bibliotecas, etc... Camada de Sistema Gerenciamento dos dados

Camada de Sistema Processamento da Aplicação

Camada de Sistema Apresentação

Cliente

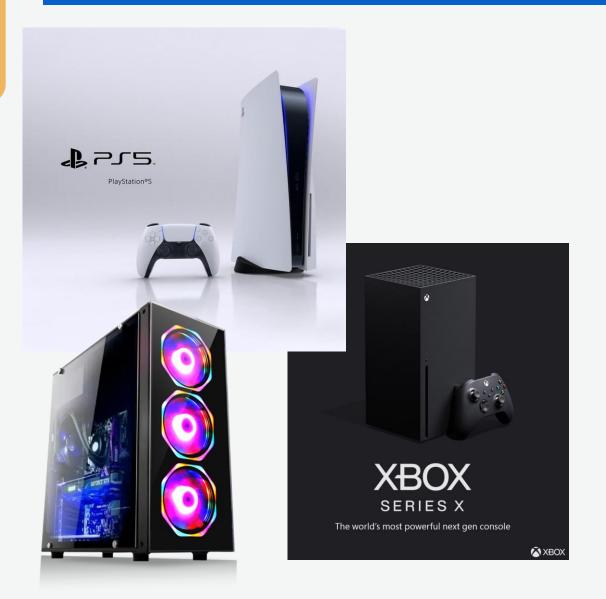
- O Hardware cliente é utilizado na sua plenitude.
- Podem ser necessárias funcionalidades apenas disponíveis em HW cliente, como acesso a periféricos, processamento diferenciado (vídeo).

<mark>Regras de negócio no servidor</mark>



 O Hardware cliente processa apenas a camada de apresentação (ex: Telas, formulários);

## Cliente Gordo vs Cliente Magro

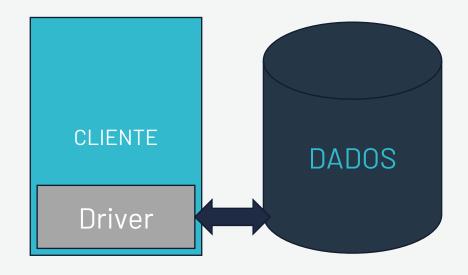






#### Modelo de Acesso a Dados [Direto]

#### Acesso direto aos dados

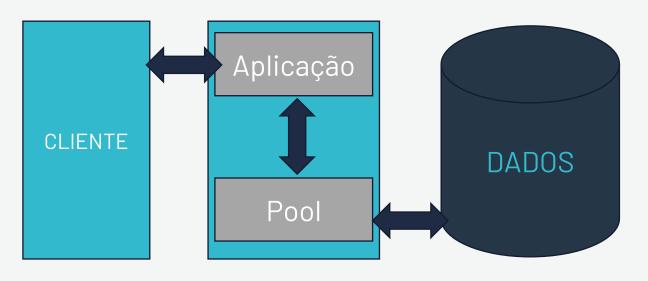


Quanto mais clientes conectados, mais consome o Banco de dados.

- A aplicação utiliza um driver e faz acesso direto ao Banco de Dados;
- A aplicação faz o controle da abertura e fechamento da conexão, ou seja, abre, utiliza e fecha.
- Aplicações "gordas" em modo cliente servidor normalmente utilizam deste modelo;
- Em aplicações antigas era comum a conexão ficar aberta durante todo o tempo;
- Pode haver muita regra de negócio no Database em formato de Stored Procedures;
- Exemplo: ODBC, ADO;

#### Modelos de Acesso a Dados [Pooling]

#### Uso de Pool de Conexões

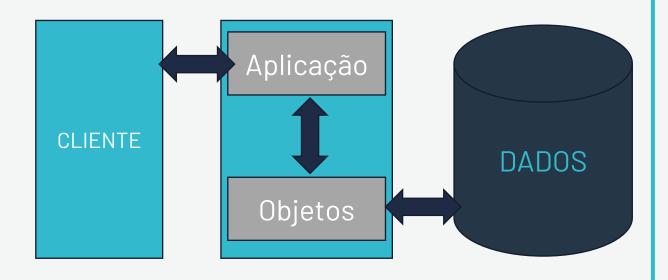


O Pool gerencia as conexões e somente ele conecta no BD. Já deixa as conexões abertas e as aplicações que solicitam o acesso já recebem uma conexão aberta.

- Existe compartilhamento das conexões e há o dimensionamento do número de conexões que estarão abertas todo o tempo e o número máximo que pode ser atingido;
- A aplicação cliente chama uma aplicação que pode ou não estar na mesma camada que utiliza uma conexão que já está aberta.
- Não há a abertura e fechamento das conexões a cada transação apesar de código existir a chamada do fechamento;
- Há economia de tempo na transação e de recursos do SGDB pois ele gerenciará menos conexões;
- As regra de negócio normalmente estão na aplicação, mas podem existir em Stored Procedures;
- Normalmente o pooling é configurado no servidor de aplicações para uso comum a todas aplicações;
- Exemplos: ADO.NET, JDBC em AppServers;

#### Modelos de Acesso a Dados [Objetos]

## Uso de Objetos que representam o Banco de Dados

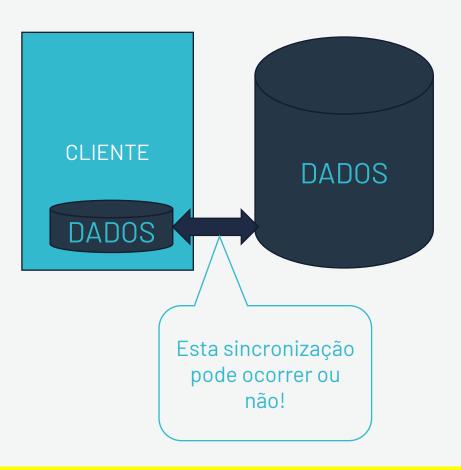


Cliente			
-codigoCliente: int		Clientes	
-nomeCliente: string		ld	Integer
+getNome(codigoCliente		Nam e	Varchar(100)

- O Banco de Dados é representado para a aplicação no formato de objetos;
- A aplicação não enxerga a estrutura de dados, ou seja, as tabelas e registros são apresentadas em formato de classes e objetos;
- O código SQL é montado automaticamente pelas bibliotecas; (é comum que problemas de performance sejam mais difíceis de serem localizados)
- Normalmente existe pooling configurado;
- A aplicação (middleware) pode fazer cache de dados em objetos o que aumenta a performance;
- Exemplos: JPA/Hibernate, Entity Framework;

#### Modelos de Acesso a Dados [Embutidos]

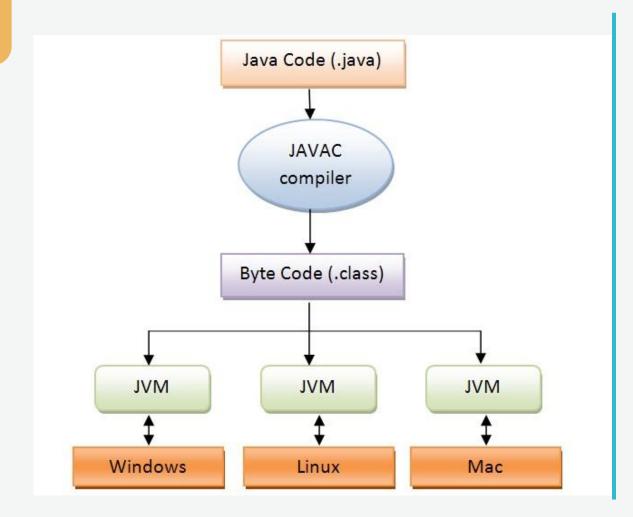
#### Banco de dados embutidos



- A aplicação se conecta em um banco de dados local;
- Utilizado para armazenar pequenas quantidades de informações e normalmente de forma temporária;
- Muito utilizado em ambientes assíncronos ou com conexão instável;
- É comum existir posterior sincronização com outras aplicações ou bases centralizadas;
- Aplicações Mobile podem se utilizar deste recurso;
- Exemplo: SQLLite; SQL Server Compact;H2;

Os smartphones utilizam. Tem uma base de dados principal mas tem uma cópia local.

# Máquinas Virtuais



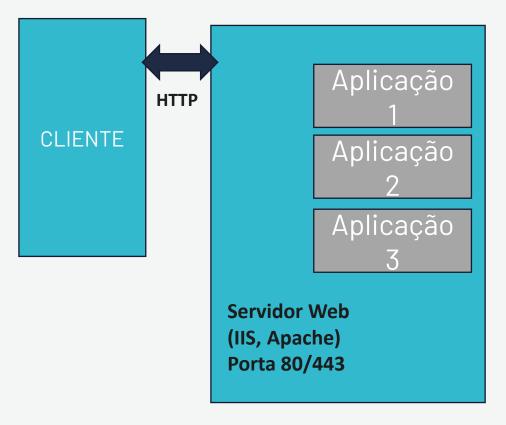
https://www.devmedia.com.br/introducao-ao-java-virtual-machine-jvm/27624

- JVM Java Virtual Machine
- O código é escrito uma vez e compilado em uma linguagem intermediária;
- A Maquina virtual instalada no sistema operacional compila ou interpreta o byte code transformando em linguagem de máquina;
- A aplicação escrita pode ser portada para diversas plataformas diferentes;
- O conceito de maquina virtual é amplamente utilizado. Ex: Java, Go, .NET

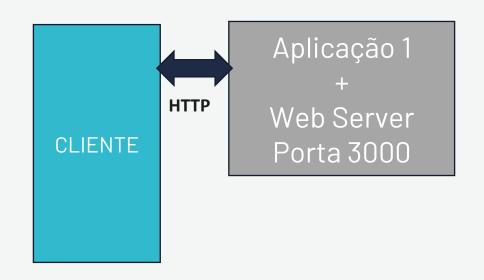
O mesmo código roda em qualquer lugar. Uma compilação roda em qualquer SO.

# Projeto com WebServer embarcado

#### Desenho tradicional. Exemplo



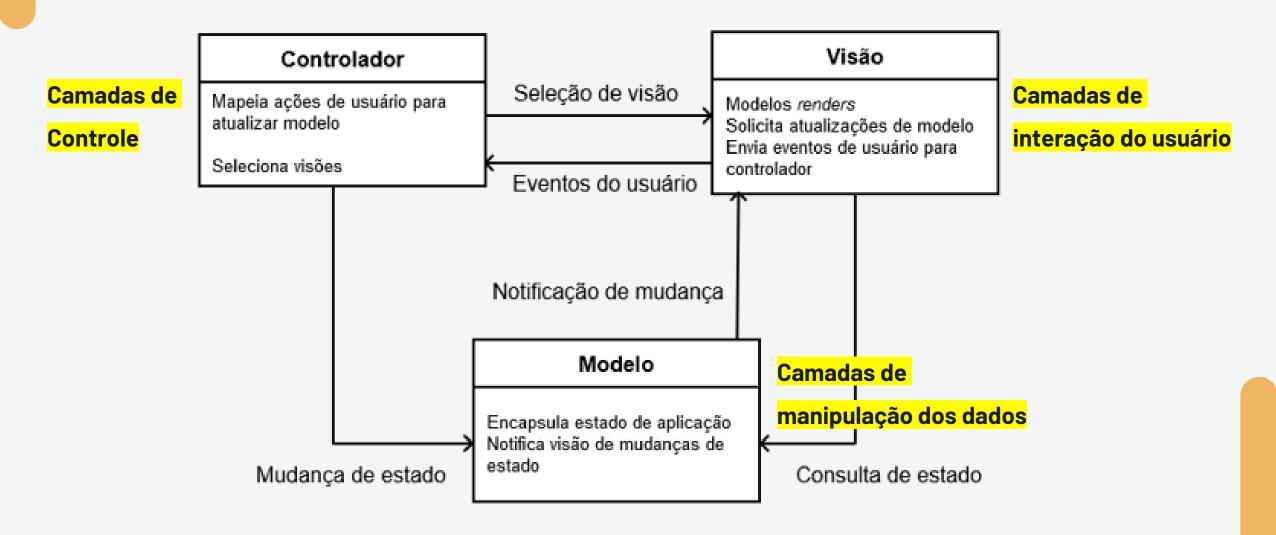
Você realiza deploy do seu projeto que é a aplicação.



Você realiza deploy do seu projeto que é a aplicação junto com um WebServer. Ex: Rails, Node.Js, SpringBoot

## O modelo MVC [Model, View, Controller]

Padrão de Arquitetura que separa a aplicação em 3 camadas: visão, controle e os dados.



# MVC [Model, View, Controller]

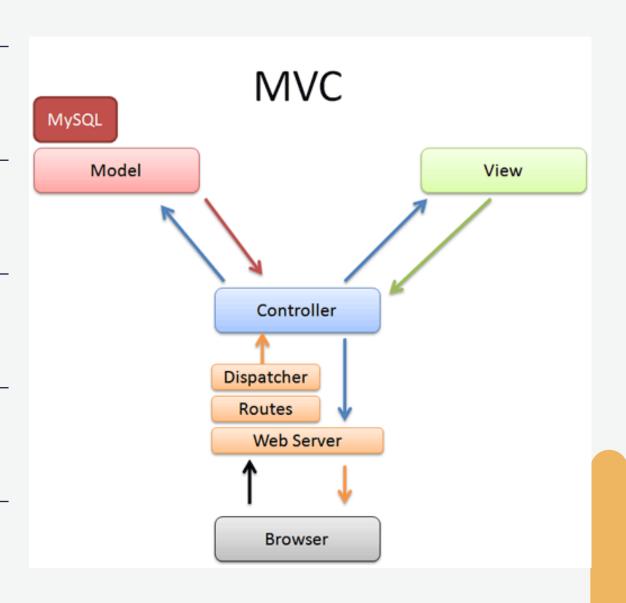
Separação da camada de apresentação da camada de controle

O controle gerencia as interações o usuário (click, tecla pressionada, ou seja, os eventos)

A camada de Modelo gerencia o acesso aos dados e como este acesso deve ser realizado

Utilizado quando há muitas formas de apresentar e interagir com os dados

Pode envolver código adicional e complexidade.



# Dicas

- 1. Mantenha os padrões de projeto consistentes em cada camada. Dentro de uma camada lógica, quando possível, o design dos componentes deve ser consistente para uma operação específica.
- 2. Não duplique a funcionalidade em um aplicativo. Deve haver apenas um componente fornecendo uma funcionalidade específica essa funcionalidade não deve ser duplicada em nenhum outro componente.
- **3. Prefira composição à herança**. Sempre que possível, use composição sobre herança ao reutilizar a funcionalidade, pois a herança aumenta a dependência entre as classes pai e filho, limitando, assim, a reutilização de classes filhas.
- 4. Estabelecer um estilo de codificação e convenção de nomenclatura para o desenvolvimento. Verifique se a organização estabeleceu padrões de estilo de codificação e nomenclatura. Se não, você deve estabelecer padrões comuns. Isso fornece um modelo consistente que torna mais fácil para os membros da equipe revisarem o código que não escreveram, o que leva a uma melhor manutenção.

# Mais Dicas

- 5. Mantenha a qualidade do sistema usando técnicas automatizadas de controle de qualidade durante o desenvolvimento. Use testes unitários e outras técnicas automatizadas de Análise de Qualidade, como análise de dependência e análise de código estático, durante o desenvolvimento. Defina métricas comportamentais e de desempenho claras para componentes e subsistemas e use ferramentas de controle de qualidade automatizadas durante o processo de criação para garantir que as decisões locais de projeto ou implementação não afetem negativamente a qualidade geral do sistema.
- 6. Considere o funcionamento de sua aplicação. Determine quais métricas e dados operacionais são exigidos pela infraestrutura de TI para garantir a implantação e operação eficientes do seu aplicativo. Projetar os componentes e subsistemas do seu aplicativo com um entendimento claro de seus requisitos operacionais individuais facilitará significativamente a implantação e a operação geral. Use ferramentas de controle de qualidade automatizadas durante o desenvolvimento para garantir que os dados operacionais corretos sejam fornecidos pelos componentes e subsistemas do seu aplicativo.

Fonte: Guia de arquitetura da Microsoft

## Diretrizes de Qualidade [Sommerville]

- O projeto deve ser exibido em um desenho de arquitetura (compreensível e que possa ser evoluído);
- 2. O projeto deve ser modular
- 3. Deve ter representações de: dados, arquitetura, interfaces e componentes
- 4. Estrutura de classes adequadas e baseadas em padrões reconhecíveis
- 5. Componentes que tenham características funcionais independentes
- 6. Deve possuir interfaces que simplifiquem a conexão entre os componentes e o ambiente externo
- 7. O projeto deve ser obtido usando um método repetível, dirigido por informações adquiridas durante a análise de requisitos de software (Aula do Prof. Alex ou Frizza)
- 8. Um projeto deve ser representado usando uma notação que efetivamente comunique seu significado.

# Reflexão: Pêndulo Infinito

# BROWSER

**Terminal de Mainframe** 

Tudo no DataCenter

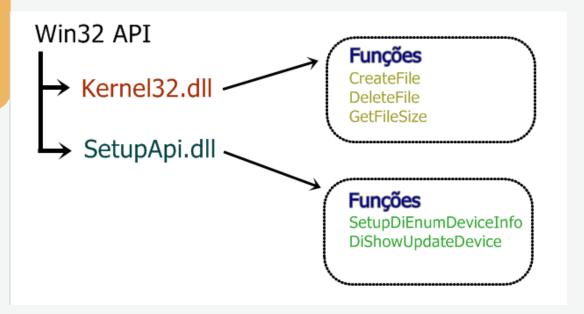


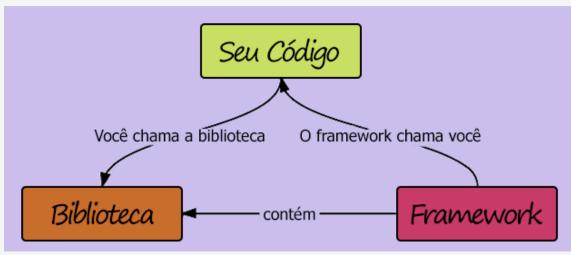
APP SMARTPHONE

> Tudo na Nuvem

As **ARQUITETURAS** estão **SEMPRE INDO E VOLTANDO**... **AS DECISÕES RARAMENTE SÃO DO TIME DE TI**...

# API, Biblioteca e Framework





API, da sigla Application Programming Interface ou Interface de Programação de Aplicações, é um produto de software criado para oferecer uma interface (caminho) com regras bem definidas para integração entre sistemas, a fim de obter informações e, assim, trabalhar com elas. É uma coleção de métodos disponibilizados para interagir com um serviço, mas sem acesso direto ao software. Serve para integrar sistemas.

Exemplo: Facebook, Windows

**Biblioteca é um conjunto de classes** (subprogramas ou funções), que podem ser usadas **para a construção de um software**. Ex: ValidaCPF, ConsultaBancoOracle.

É como um canivete suíço, VOCÊ USA o que precisa.

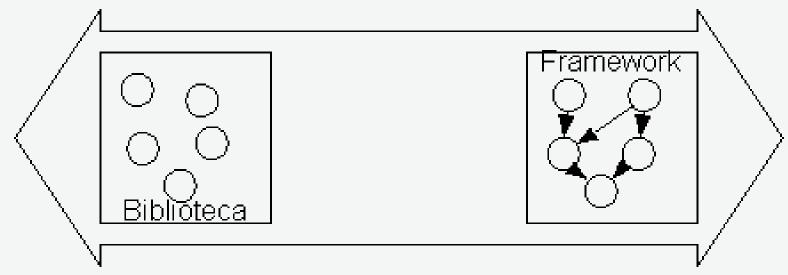
#### Framework

Um framework captura a funcionalidade comum a várias aplicações.

Um framework pode ser construído utilizando-se diversas bibliotecas e integrado a APIs. O framework é um conjunto de classes, bibliotecas e códigos que colaboram entre si.

https://pt.stackoverflow.com/questions/17501/qual-%C3%A9-a-diferen%C3%A7a-de-api-biblioteca-e-framework

# API, Biblioteca e Framework



- Classes instanciadas pelo cliente.
- Cliente chama funções
- Não tem fluxo de controle predefinido
- Não tem interação predefinida
- Não tem comportamento default

- Customização com subclasse ou composição
- Chama funções da "aplicação"
- Controla o fluxo de execução
- Define interação entre objetos
- Provê comportamento default

http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/oque.htm

Framework é uma estrutura genérica que pode ser ampliada para criar um subsistema ou aplicação mais específica. (Sommerville). O framework pode ser ampliado e pode ser necessária a adição de novas classes que vão herdar funcionalidades das classes abstratas.



#### **BIBLIOTECA**

Uma biblioteca se refere a uma coleção de pacotes que fornecem funções. Seu objetivo é oferecer um conjunto de funcionalidades prontas para uso sem se preocupar com outros pacotes. Como em uma biblioteca de livros, você precisa buscar o conhecimento (funcionalidade).

#### **Exemplos:**

- **Moment.js**: Biblioteca para converter, validar, manipular e exibir datas e horários.
- **Chart.js**: Biblioteca para criação de gráficos.
- mo.js: Biblioteca para criar animações com SVG.
- **React**: Biblioteca para criar interfaces de usuário.

#### **FRAMEWORK**

Conjunto de bibliotecas, classes e códigos. Um framework não oferece apenas funcionalidades, mas também uma arquitetura para o trabalho de desenvolvimento. Você **não** cria ou inclui uma estrutura, em um framework, você integra seu código a ele.

#### **Exemplos:**

- Angular: Framework para criação de aplicações web
- Vue.js: Framework para criação de aplicações web
- **lonic**: Framework para criar arquivos mobile
- Express: Framework para criar aplicações com Node.js

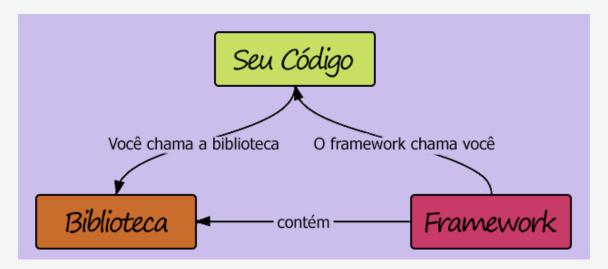
# Frameworks

Frameworks são ferramentas, não modos de vida. "Não case com o Framework!"

A arquitetura deve falar sobre o sistema e não sobre os frameworks que você utilizou no sistema.

Evite que os frameworks entrem no seu código central do sistema.

(Polêmico, mas muito seguido por Empresas feitas para durar)



# Toolkit e SDK

**Toolkits** funcionam de forma mais livre, não são frameworks, similares a bibliotecas funcionando em conjunto, ou seja, você usa o que precisa.

#### Kit de ferramentas para executar determinada atividade.

Ex: Google Web Toolkit. Conversão de classes java para javascript.

**SDKs. Software Development Kits** podem ser toolkits ou frameworks. Contém ferramentas adicionais além das bibliotecas e documentações. Podem vir com exemplos de códigos que ajudam a usar a biblioteca adequadamente.

#### Kit de ferramentas para desenvolvimento de um determinado software.

Ex: Java SDK (compilador, bibliotecas, utilitários)

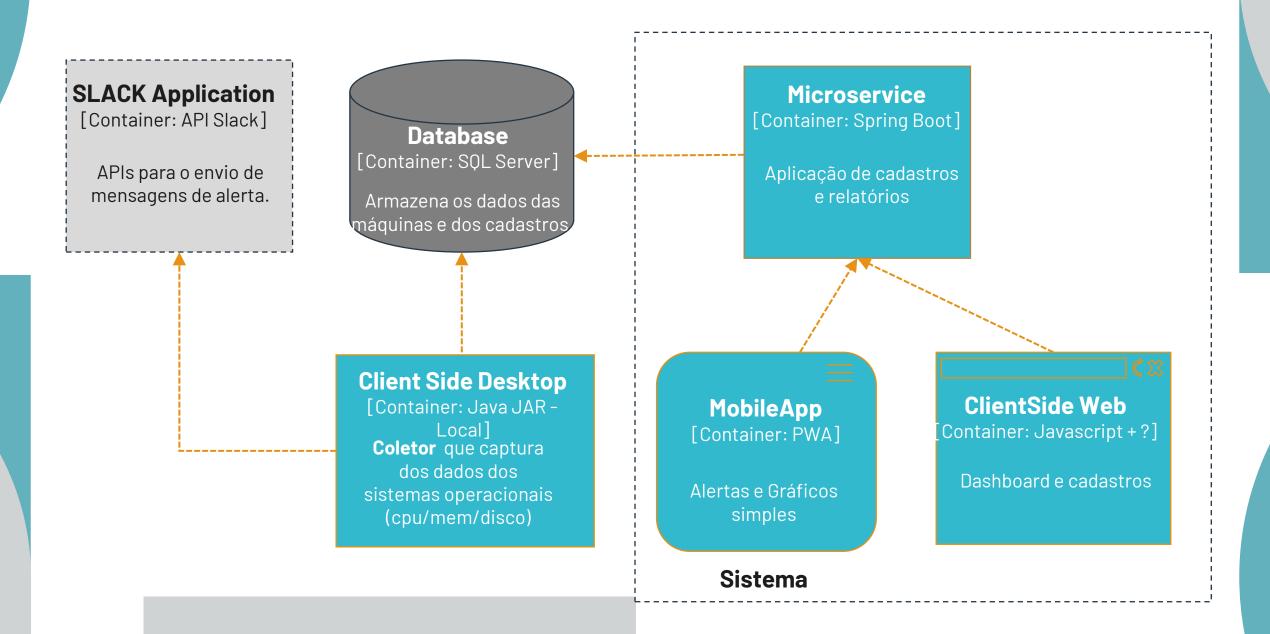
# Entregável de PI:

# Diagrama de Solução de

# Software

[Desenho de Arquitetura]

## Diagrama – Visão – Containers



# Conceitos que serão utilizados

Vamos pensar em containers (<u>não é Docker</u>), mas pensar que o **container é conjunto que precisa estar funcionando ou rodando para um software funcionar**.

Exemplos de Containers (Representados por grandes quadrados):

Server-side web application: Aplicação backend. Ex: Spring MVC, NodeJs, Asp.NET MVC, etc.

Client-side web application: A aplicação Javascript que roda no Web Browser. Ex: Angular, JQuery,

React.

Client-side desktop application: A aplicação que roda local. Ex: Java JAR, .NET Windows, C++.

**Mobile app**: Ex: App IOS, App Android, App React Native.

**Server-side console application**: Ex: "public static void main" application, batch, script.

**Microservice**: Ex: Spring Boot.

**Serverless function**: Uma função que independe se servidor. Ex: Amazon Lambda, Azure Function.

Database: Um banco de dados relacional ou de objetos. Ex: MySQL, SQL Server, Oracle Database,

MongoDB.C

# Passo a passo - Diagrama de Solução de Software (Entregável de PI)

- 1. Identificar os Objetivos das Arquitetura
- 2. Cenários Chave
  - 1. O que é crítico para o negócio?
  - 2. O que gera alto impacto?
- 3. Fazer a visão global (overview) da Aplicação
  - 1. Determinar o tipo da sua aplicação (WEB, Mobile, etc)
  - 2. Identificar as restrições no desenvolvimento (Rede, Segurança, Sistema Operacional)
  - 3. Identificar estilos importantes de arquitetura (Camadas, SOA) Vamos ver mais a frente.
  - 4. Determinar as tecnologias relevantes (Spring, Node.JS)
- 4. Desenhar no slide utilizando os modelos de contêineres disponíveis.
- Identificar os assuntos chaves (Key Issues: Qualidade, Deploy, Execução, Usabilidade)
- 6. Cuidar dos itens Transversais (Caching, Comunicação, Autenticação, etc).



# Agradeço a sua atenção!

Fábio Figueredo

fabio.figueredo@sptech.school



SÃO PAULO TECH SCHOOL