

# Principais Arquiteturas de Software da Atualidade

CAPÍTULO 1. Arquitetura de Sistemas Web

PROF. Albert Tanure

# Principais Arquiteturas de Software da Atualidade

---

Aula 1.1. Introdução

PROF. Albert Tanure

# Nesta aula



- ☐ O que vamos estudar?
- ☐ Pré-requisitos
- ☐ O que é design e arquitetura?

# O que vamos estudar?



Alguns modelos arquiteturais existentes

A diagram on the left side of the slide shows a blue circle representing a person's head. Three callout boxes, shaped like speech bubbles or sticky notes, point towards the person. The top box is teal and contains the text 'Alguns modelos arquiteturais existentes'. The middle box is dark blue and contains the text 'Exemplos de implementação'. The bottom box is teal and contains the text 'NÃO TEREMOS FÓRMULAS MÁGICAS'. The background of the slide is white with a large, light gray circular shape on the left and a teal wavy shape at the top right.

Exemplos de implementação

NÃO TEREMOS FÓRMULAS MÁGICAS

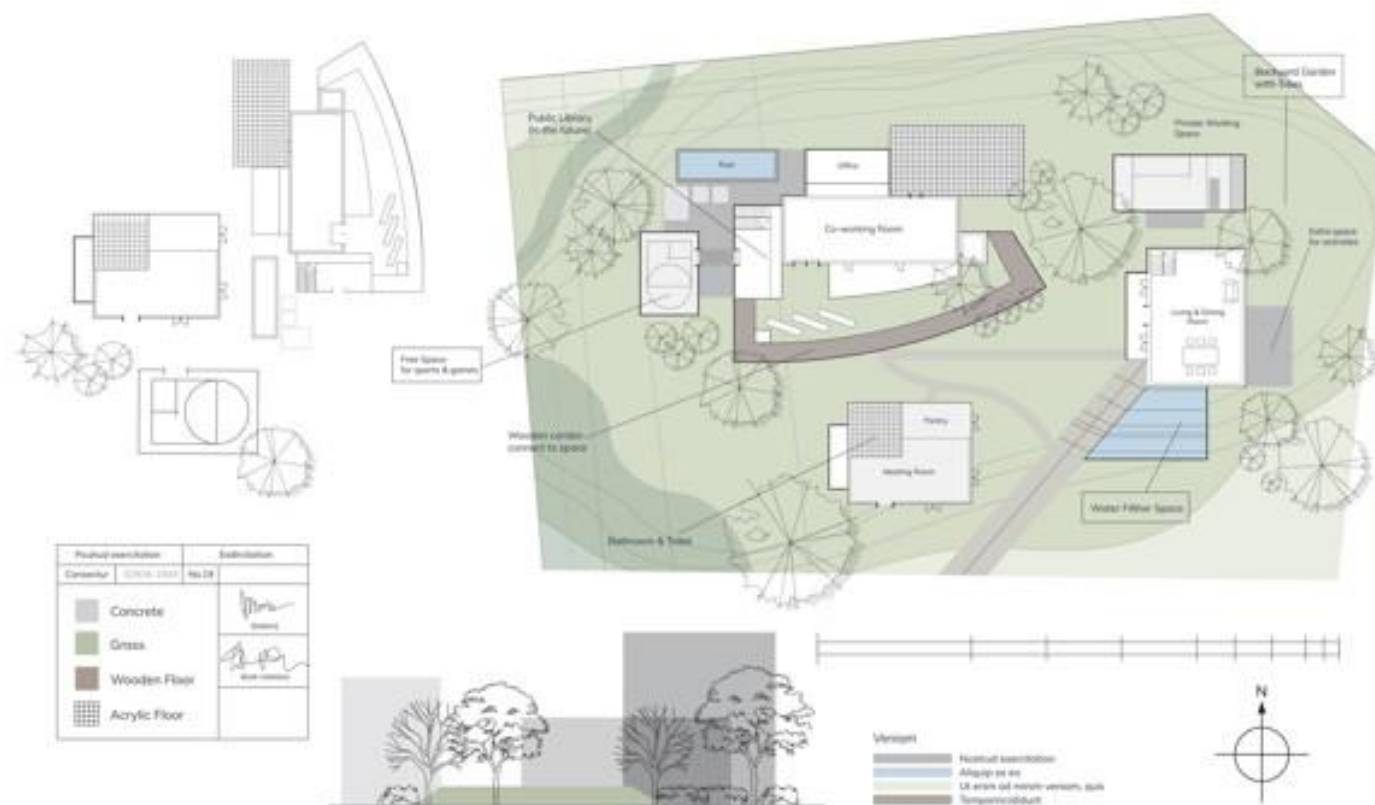
# Pré-requisitos

- Ambiente de desenvolvimento (java, .net, etc)
- Editor de códigos

# O que é design e arquitetura?



Uncle Bob




*“The goal of software architecture is to minimize the human resources required to build and maintain the required system.”*

Uncle bob – Cleand Architecture






IGTi

An abstract graphic element consisting of two overlapping, rounded shapes. The top shape is teal and the bottom shape is purple. They are both slightly irregular and have a soft, organic feel.

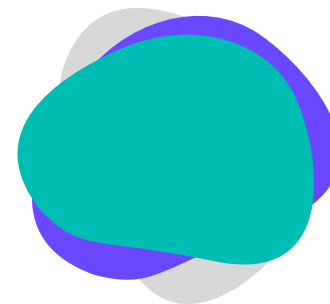
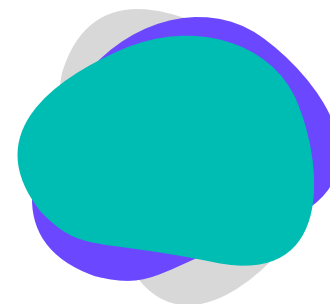
As decisões de hoje podem afetar sua  
arquitetura no futuro.

Não se baseie na moda

An abstract graphic element consisting of two overlapping, rounded shapes. The top shape is teal and the bottom shape is purple. They are both slightly irregular and have a soft, organic feel.

Padrões devem ser analisados, estudados,  
serem base, mas faça adaptações à sua  
realidade

Trabalhe de forma evolutiva



# Conclusão

- A arquitetura de software é uma arte
- Mais do que classes e objetos suas decisões devem ser baseadas em dados, não em preferências
- Seja disponível, humilde e crie um ambiente colaborativo
- Oriente o seu pensamento para entrega de valor e evolua constantemente sua arquitetura

# Próxima aula



01.

Características de aplicações web modernas

02.

Modularização, Testes

03.

Web, SPA, Cross-platform

04.

Cloud

# Principais Arquiteturas de Software da Atualidade

---

Aula 1.2. Características de aplicações web modernas

PROF. Albert Tanure

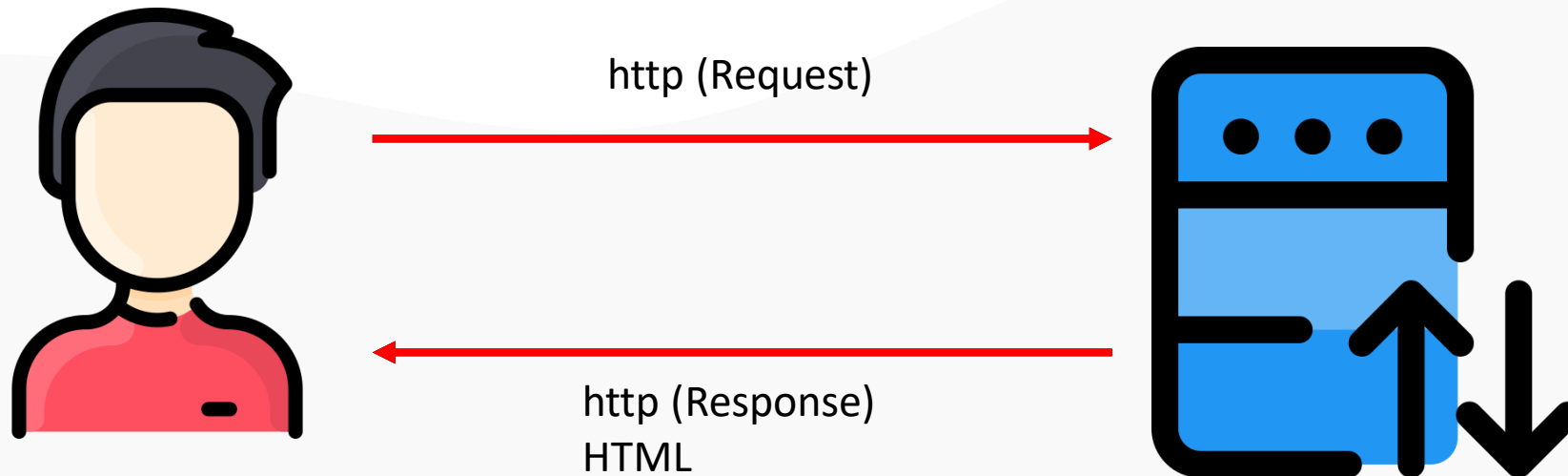
# Nesta aula



- ☐ Principais características de aplicações web
- ☐ Desafios

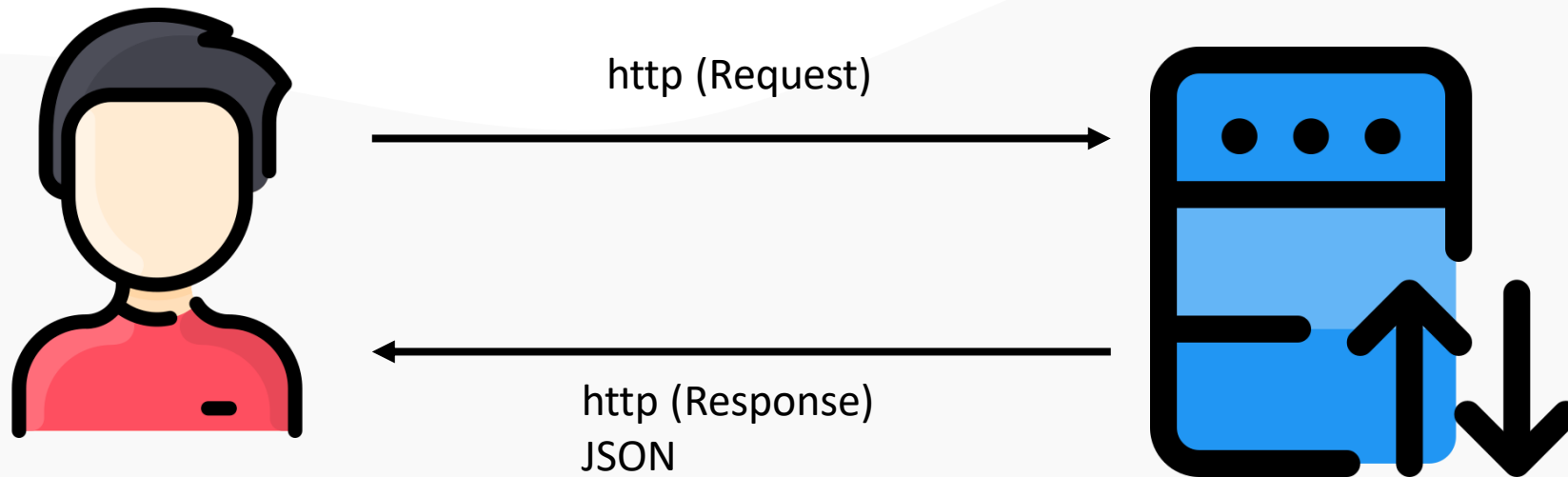
# Características de aplicações web

# Client-Server





# SPA



# Desafios

# Mundo Conectado

IGTi



# HTML & JavaScript



```
lines (22 sloc) | 729 Bytes
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://apple.com/DTD/PLIST-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>CFBundleDevelopmentRegion</key>
    <string>Corona App iOS</string>
    <key>CFBundleExecutable</key>
    <string>$(EXECUTABLE_NAME)</string>
    <key>CFBundleIdentifier</key>
    <string>$(PRODUCT_BUNDLE_IDENTIFIER)</string>
    <key>CFBundleInfoDictionaryVersion</key>
    <string>6.0</string>
    <key>CFBundleName</key>
    <string>$(PRODUCT_NAME)</string>
    <key>CFBundlePackageType</key>
    <string>$(PRODUCT_PACKAGE_TYPE)</string>
```

```
describe('Testing Cypress.io', () => {
  before(() => {
    cy.visit('https://cypress.io');
  });

  it('Closing banners should close banners', () => {
    // Testing top banner
    cy.get('.close-top-banner-btn')
      .should('be.visible')
      .click()
      .should('not.exist');

    // Testing cookie consent
    cy.get('.cookieConsent').should('be.visible');
  });
});
```

# Demanda

IGTI



# Cloud

igti

Azure

GCP

AWS

# Escalabilidade

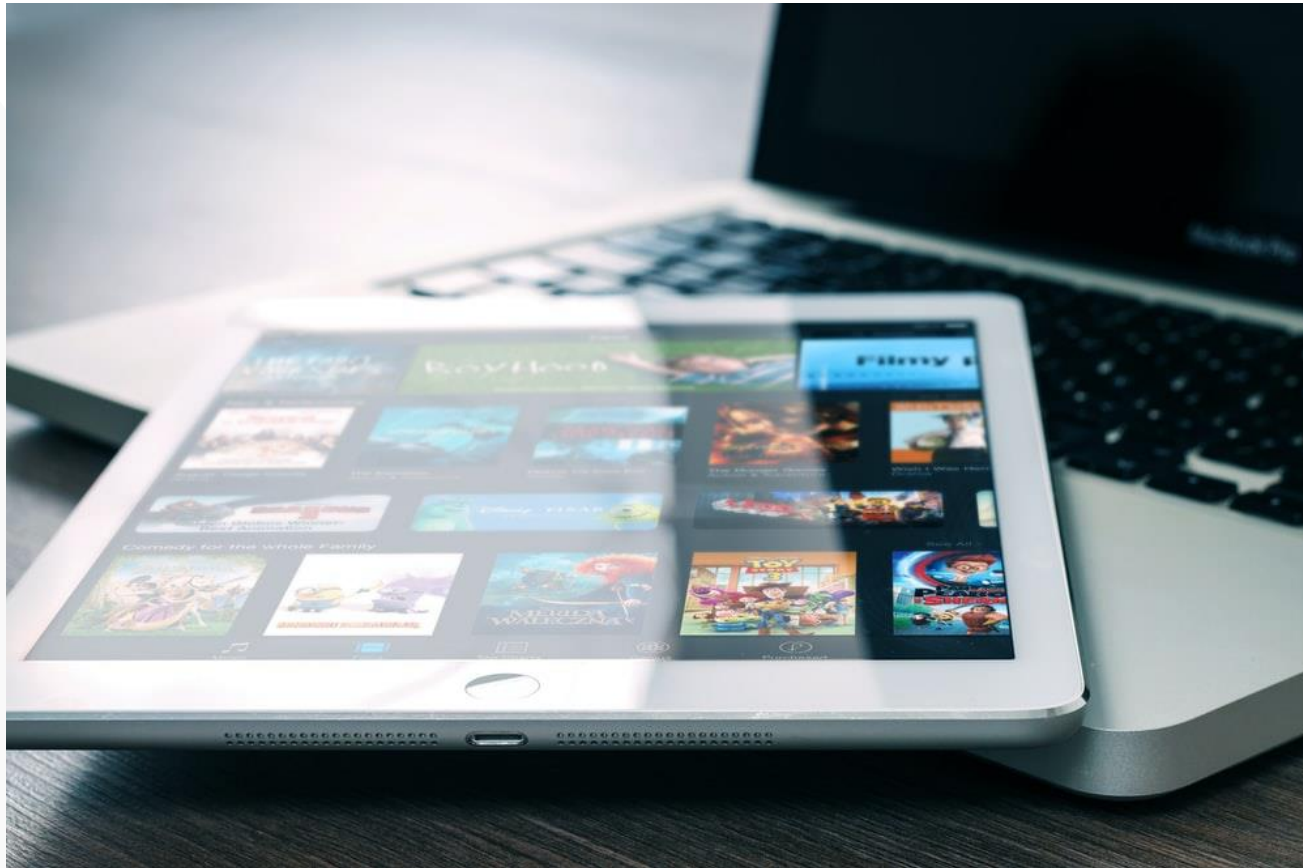
IGTi





# Devices

IGTI





# Testes

IGTi



# Monitoramento



# Desafios



01.

Características de aplicações web modernas

02.

Modularização, Testes

03.

Client-Server, SPA, Cross-platform

04.

Cloud

# Conclusão

- Aplicações web vão além de um simples site
- A demanda do mercado tem aumentado a complexidade para construir e gerenciar aplicações para diversas plataformas
- Um mundo além do JavaScript e do HTML

# Próxima aula



01.

Web applications (Client-Server)

02.

Tecnologias

03.

SPA

04.

Estratégias

# Principais Arquiteturas de Software da Atualidade

---

Aula 1.3. Web App

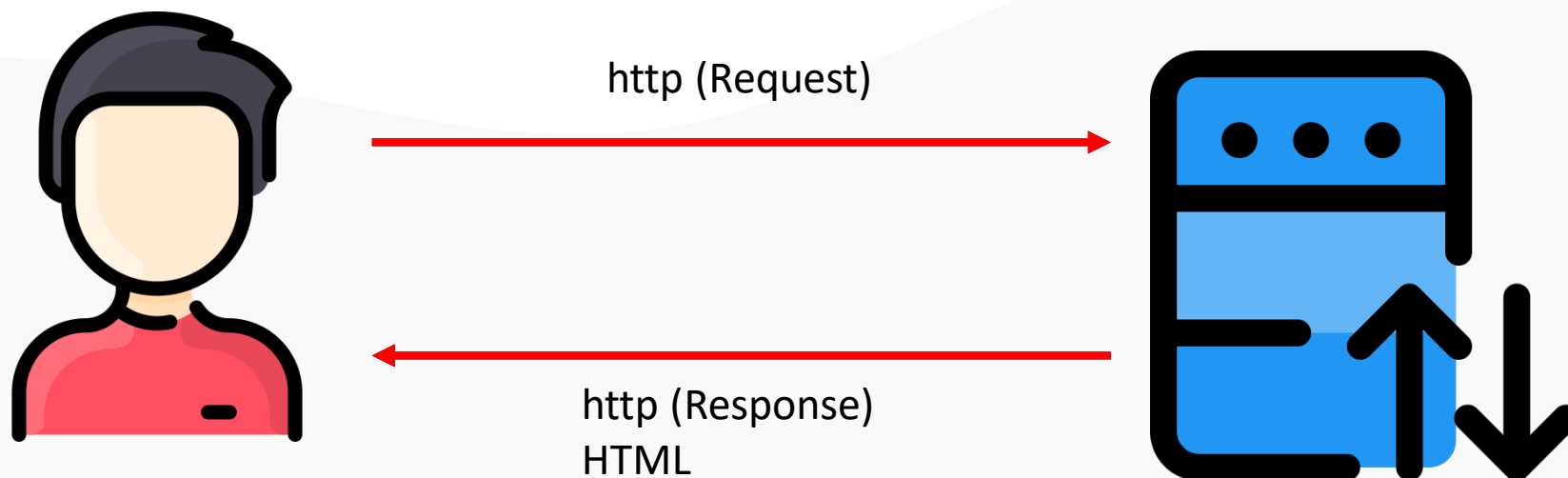
PROF. Albert Tanure

# Nesta aula

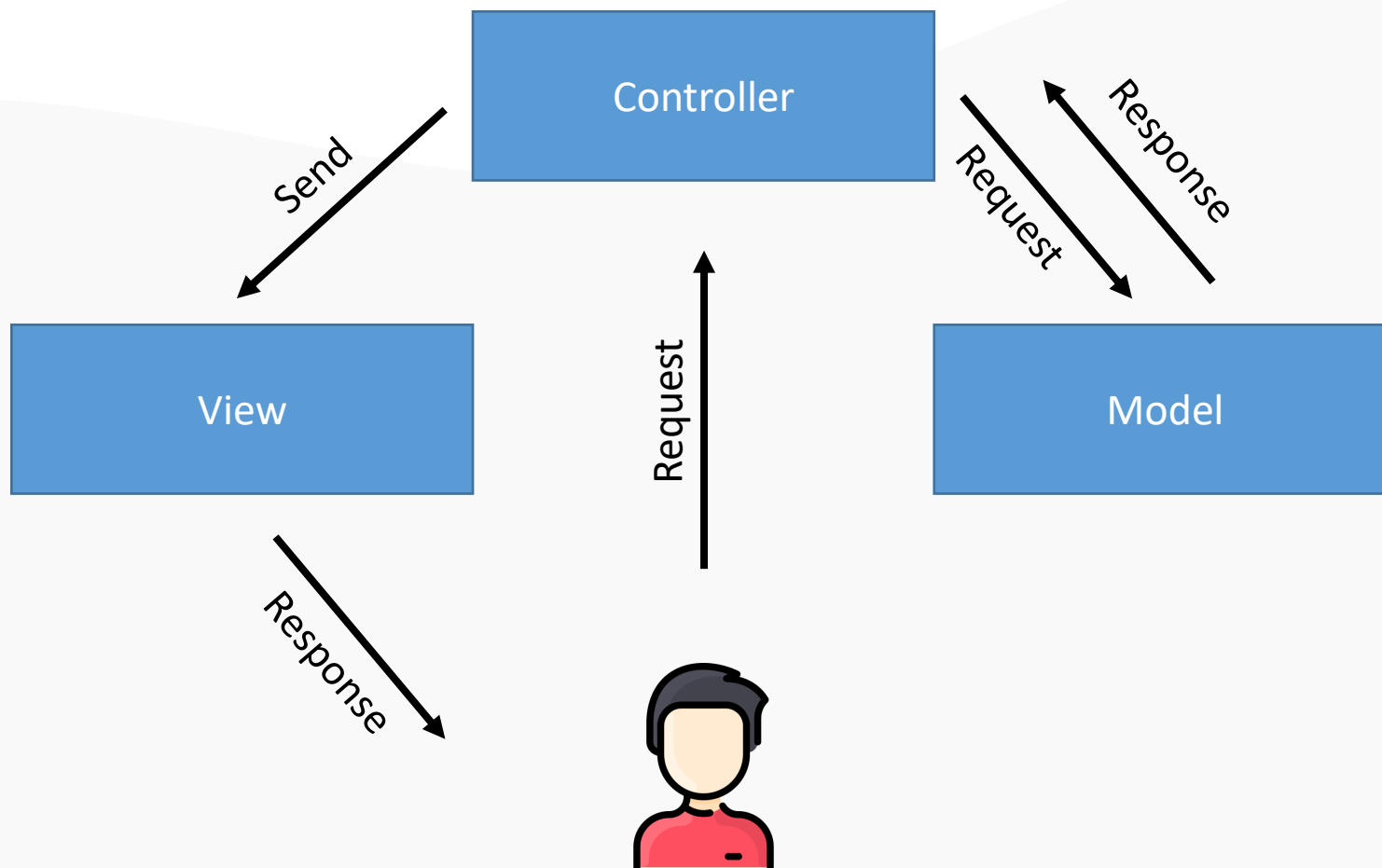


- ☐ Client-server
- ☐ Demo

# Client-Server







# Demo

# Conclusão

- Client-server são aplicações robustas
- Depende de uma boa organização de estrutura e código
- Demanda maior controle no servidor
- Mais difícil de implementar uma solução Cross-Platform

# Próxima aula



01.

SPA

02.

Arquitetura

03.

Conceito

04.

Demonstração

# Principais Arquiteturas de Software da Atualidade

---

Aula 1.4. SPA

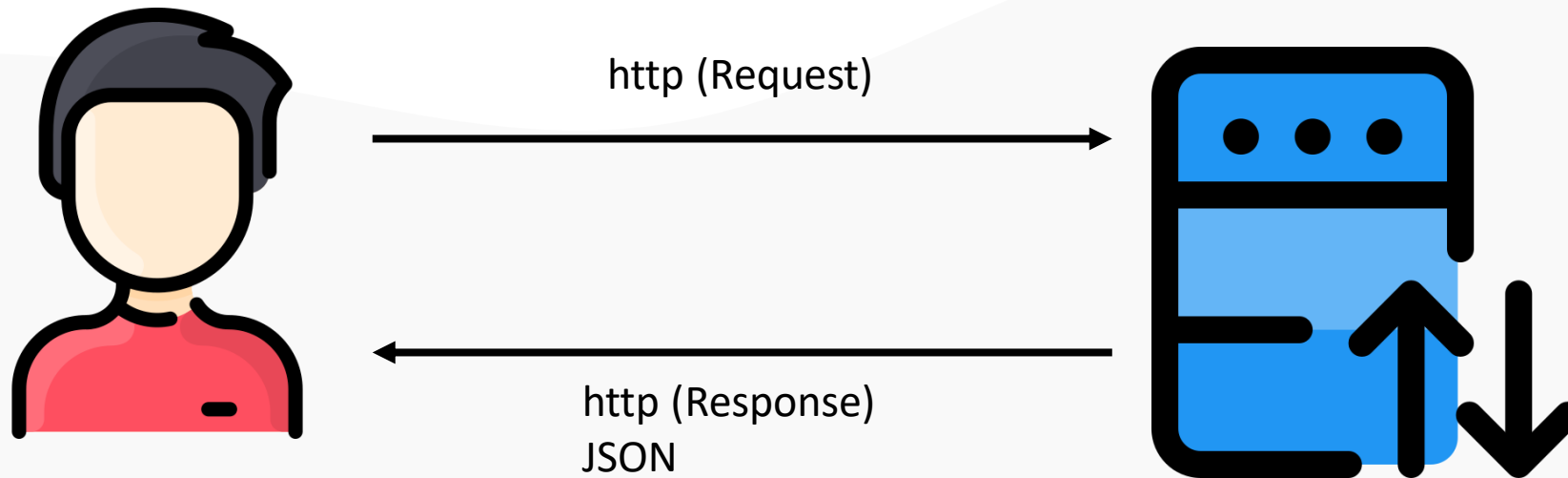
PROF. Albert Tanure

# Nesta aula



- ☐ SPA
- ☐ Características
- ☐ Gerenciamento de pacotes
- ☐ Tecnologias
- ☐ Como escolher um framework SPA?

# SPA



# Características



- HTML
- JavaScript
- CSS
- CSS – Pré-processadores



# Gerenciamento de Pacotes



HTML

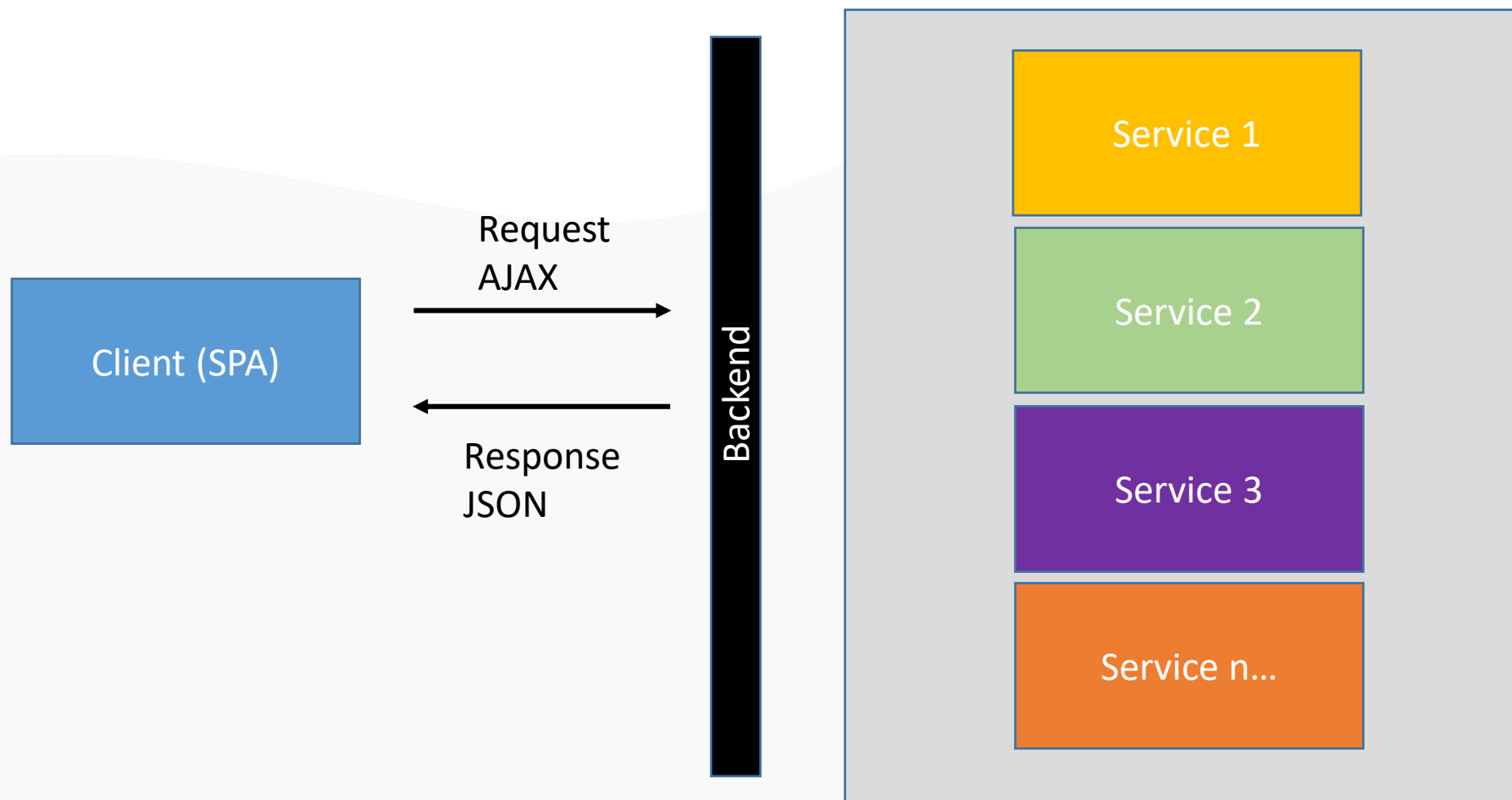
```
<!-- development version, includes helpful console warnings -->  
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

Fonte: <https://docs.microsoft.com/pt-br/dotnet/architecture/modern-web-apps-azure/common-client-side-web-technologies>



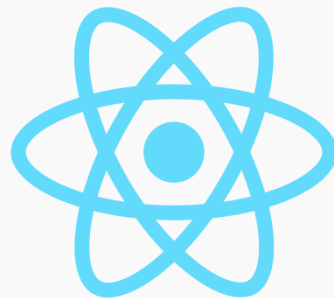
<https://www.npmjs.com/>

Fonte: <https://icons8.com/icon/24895/npm>



# Tecnologias

IGTi



# Arquitetura



Interceptadores

Resources

Notifications

Exceptions

Componentes

Storage

APIs

Testes

# Como escolher um framework SPA?



- Sua equipe está familiarizada com esta abordagem?
- Concorda com a abordagem de implementação do Framework?
- O Framework inclui todos os recursos necessários para seu aplicativo?
- Está bem documentado?
- Quão ativa é sua comunidade? Novos projetos estão sendo compilados com ele?
- Quão ativa é o repositório do framework? Os problemas estão sendo resolvidos e novas versões são fornecidas regularmente?

# Conclusão



- SPA
- Tecnologias e benefícios
- Diversidade de frameworks
- Permite a implementação de boas práticas arquiteturais

# Próxima aula



01.

Princípios arquiteturais

02.

Padrões

03.

Estilos

04.

Exemplos

# Principais Arquiteturas de Software da Atualidade

---

Aula 1.5. Princípios Arquiteturais

PROF. Albert Tanure

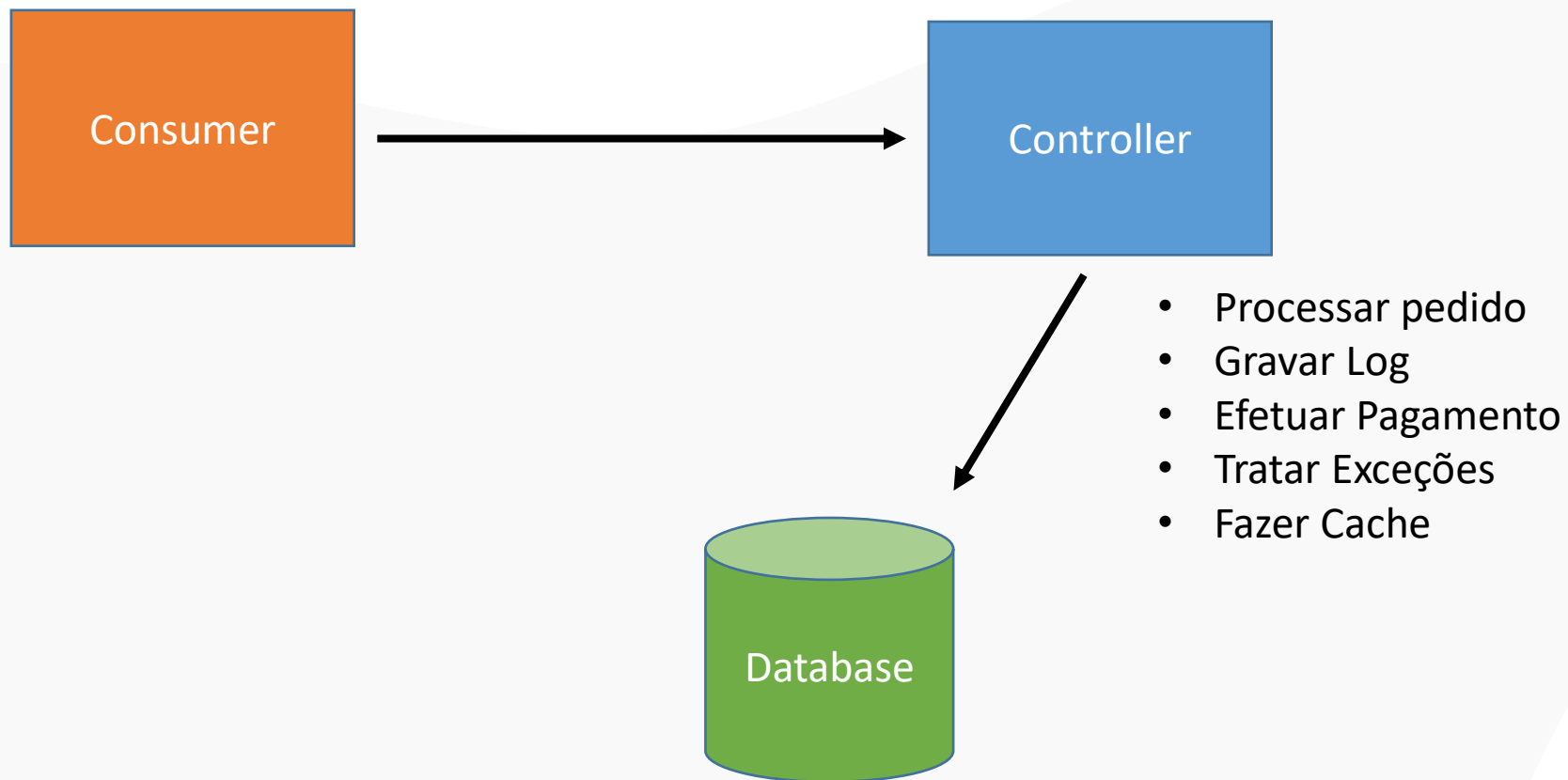


# Nesta aula

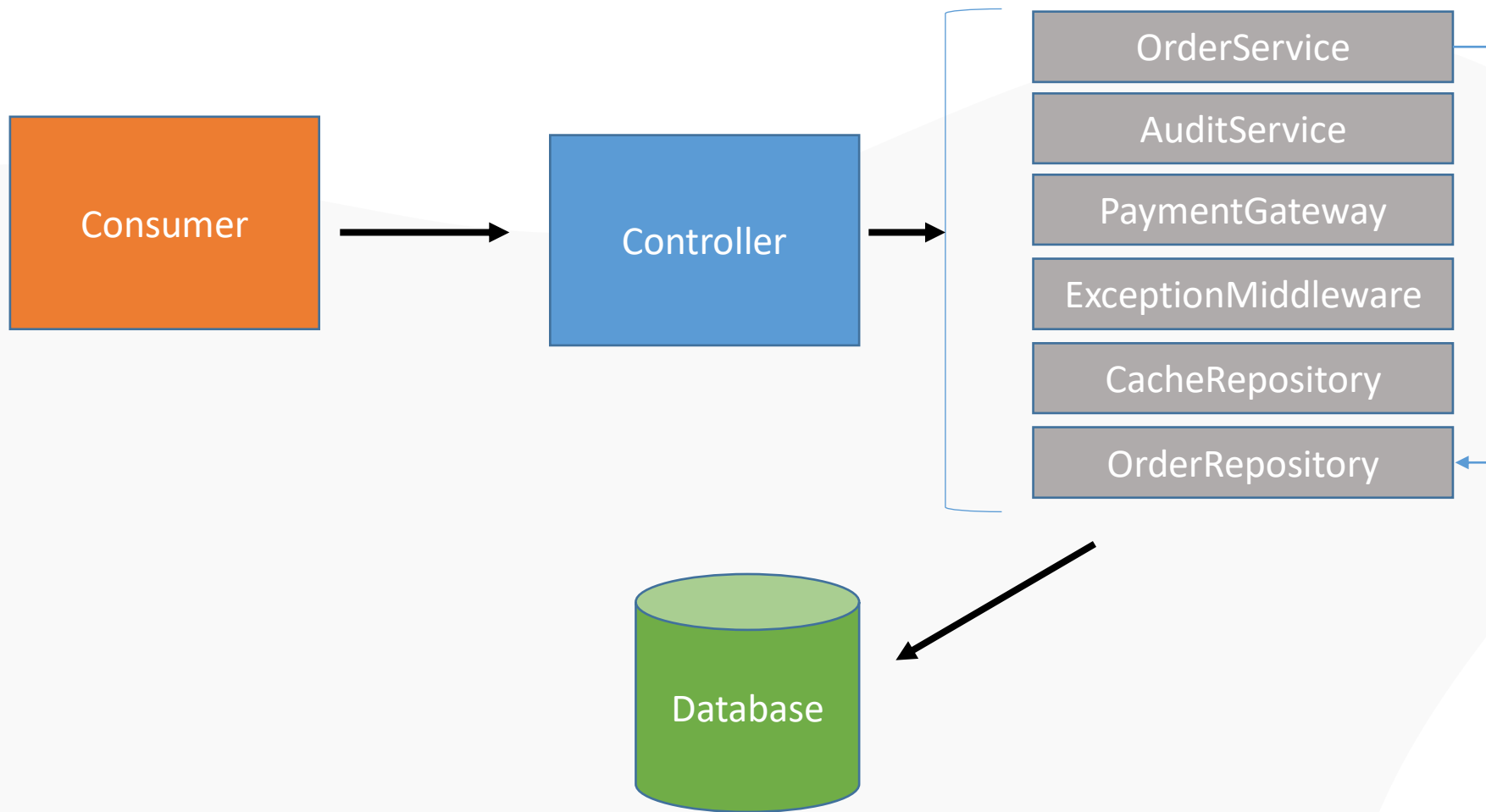


- ☐ Separação de responsabilidades
- ☐ Encapsulamento
- ☐ Injeção de dependência
- ☐ Dependências explícitas
- ☐ Responsabilidade única
- ☐ Don't repeat yourself (DRY)
- ☐ Ignorância de Persistência

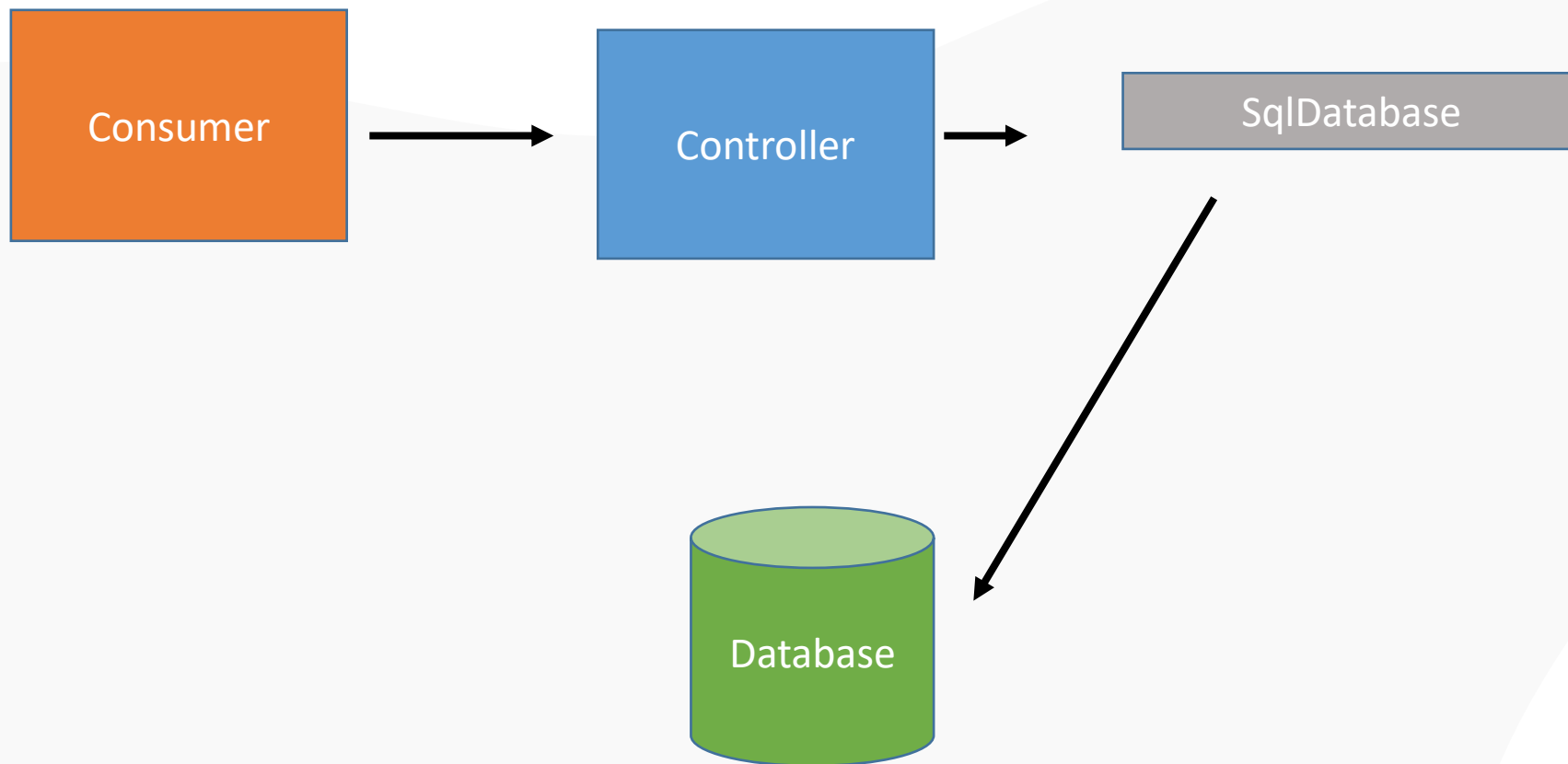
# Separação de responsabilidades



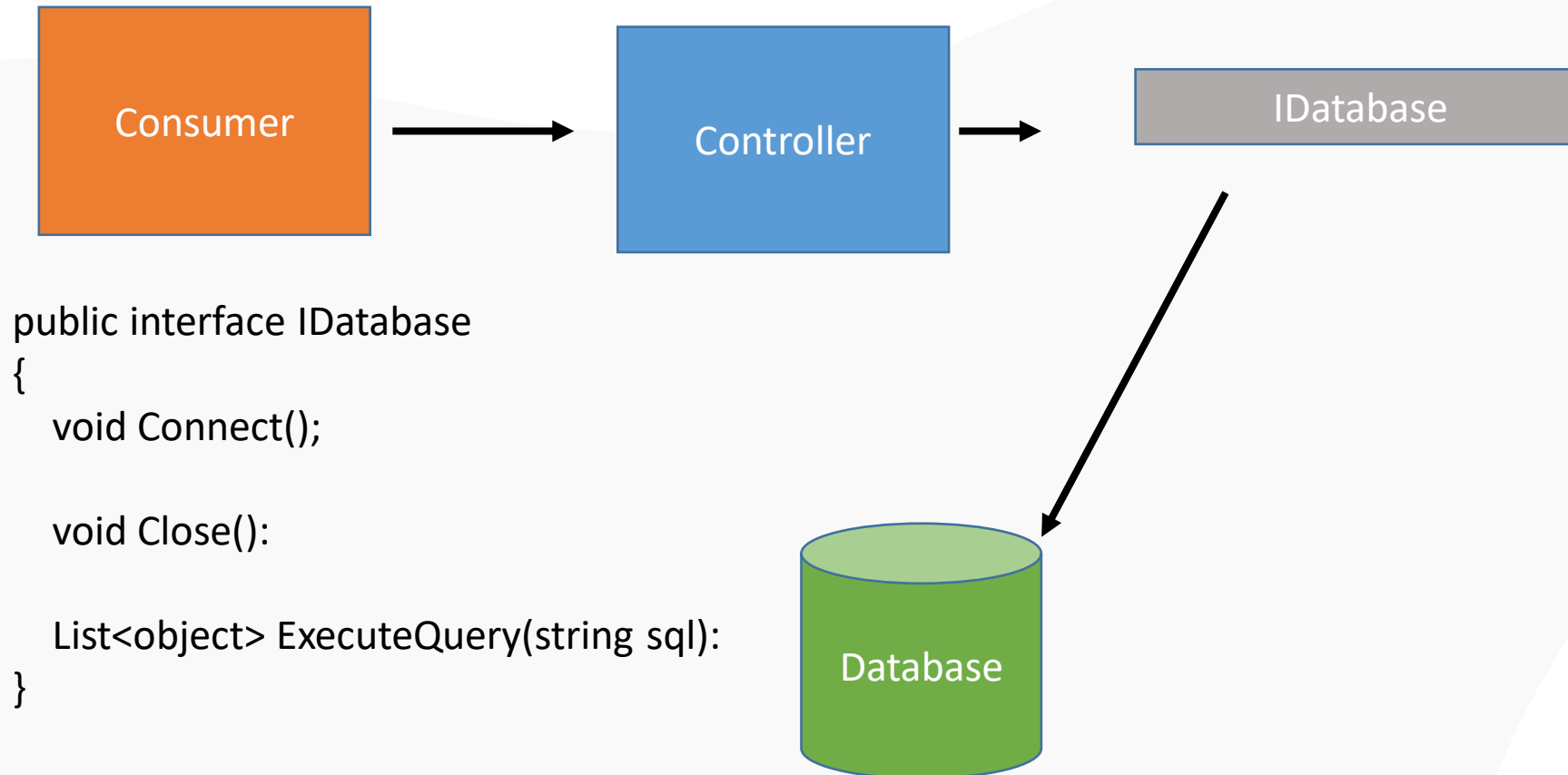
# Separação de responsabilidades



# Separação de responsabilidades



# Separação de responsabilidades



# Encapsulamento



```
public class Pessoa
{
    public string Nome { get; set; }
```

```
    public int Idade { get; set; }
}
```

```
public class Pessoa
{
    public string Nome { get; private set; }
```

```
    public int Idade { get; private set; }
```

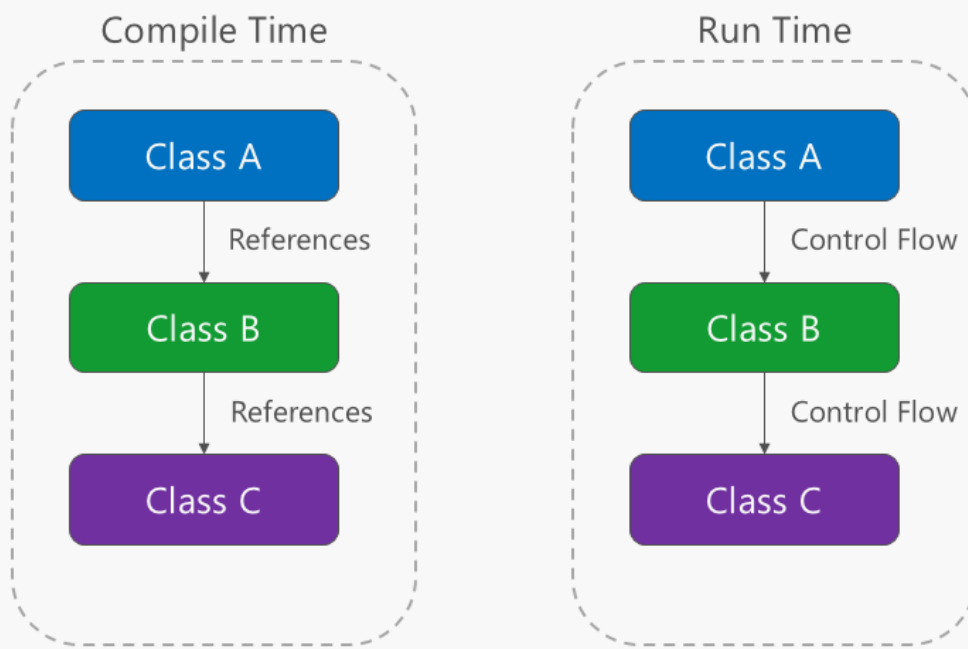
```
    public Pessoa(string nome, int idade)
    {
        Nome = nome;
        Idade = idade;
    }
}
```

```
Pessoa pessoa = new Pessoa();
p.Nome = "Albert";
p.Idade = 39;
```

```
p.Idade = 20;
```

# Injeção de dependência

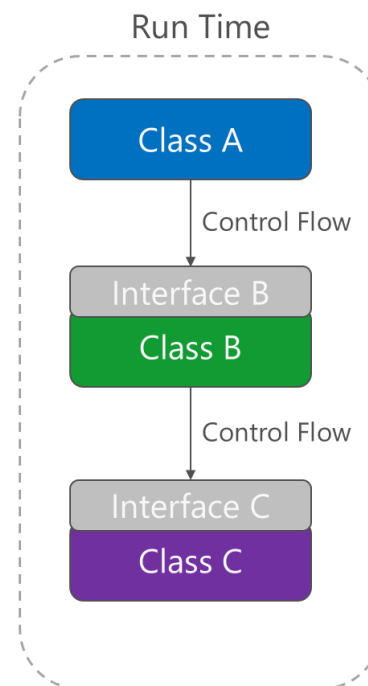
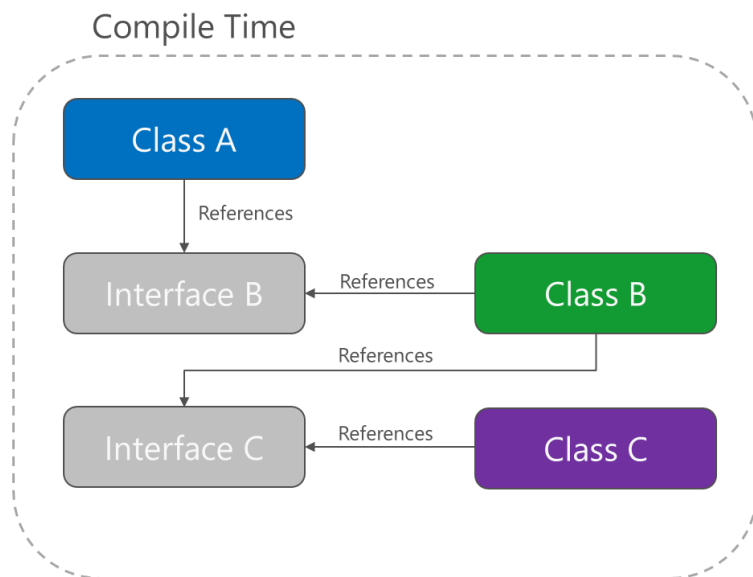
## Direct Dependency Graph



# Injeção de dependência



## Inverted Dependency Graph





# Dependências Explícitas



```
public class OrderService
{
    public OrderService(IOrderRepository repository, ILog log)
    {
        // Comands
    }
}
```

# Responsabilidade Única



```
public class OrderService
{
    public OrderService(IOrderRepository repository, ILog log)
    {
        // Comands
    }

    pulic void Save(...) {...}

    public void SetCache(...) {...}

    public Stream GeneratePDF(...) {...}

    public void SendEmail(...) {...}
}
```

# Don't Repeat Yourself (DRY)



```
public ClientViewModel
{
    public string FirstName { get; private set; }

    public string LastName { get; private set; }

    public string FullName
    {
        get { return $"{Name} {LastName}"; }
    }

    public ClientViewModel(Client client)
    {
        FirstName = client.Name;
        LastName = client.LastName;
    }
}
```

```
public DependentViewModel
{
    public string FirstName { get; private set; }

    public string LastName { get; private set; }

    public string FullName
    {
        get { return $"{Name} {LastName}"; }
    }

    public DependentViewModel(Client client)
    {
        FirstName = client.Name;
        LastName = client.LastName;
    }
}
```

# Don't Repeat Yourself (DRY)



```
public class NameValueObject
{
    public string FirstName { get; private set; }

    public string LastName { get; private set; }

    public string FullName
    {
        get { return $"{Name} {LastName}"; }
    }

    public NameValueObject(Client client)
    {
        FirstName = client.Name;
        LastName = client.LastName;
    }
}
```

```
public DependentViewModel
{
    public NameValueObject Name { get; private set; }

    public DependentViewModel(Client client)
    {
        Name = new NameValueObject(client);
    }
}

public ClientViewModel
{
    public ClientViewModel Name { get; private set; }

    public DependentViewModel(Client client)
    {
        Name = new NameValueObject(client);
    }
}
```

# Ignorância de Persistência

- Refere-se aos tipos que precisam ser persistidos, cujo código não é afetado pela opção de tecnologia de persistência
- Permite utilização de vários modelos de persistência
- Facilita a utilização de técnicas como mapeamento de domínio e desenvolvimento orientado a testes

# Conclusão

- O conhecimento de fundamentos é essencial para desenhar boas soluções
- Exercite a revisão de códigos(Code Review) com o time
- No fim, o bom e velho paradigma Orientado a Objetos nos direciona para melhores desenhos
- Divida para conquistar

# Próxima aula



01.

Principais Arquiteturas

02.

Onion Architecture

03.

Clean Architecture

04.

Hexagonal Architecture

# Principais Arquiteturas de Software da Atualidade

---

Aula 1.6. Principais Arquiteturas

PROF. Albert Tanure



# Nesta aula



- ☐ Arquitetura em Camadas
- ☐ Arquitetura Monolítica
- ☐ Onion Architecture / Clean Architecture
- ☐ Hexagonal Architecture
- ☐ Arquitetura orientada a eventos

# Arquitetura em camadas

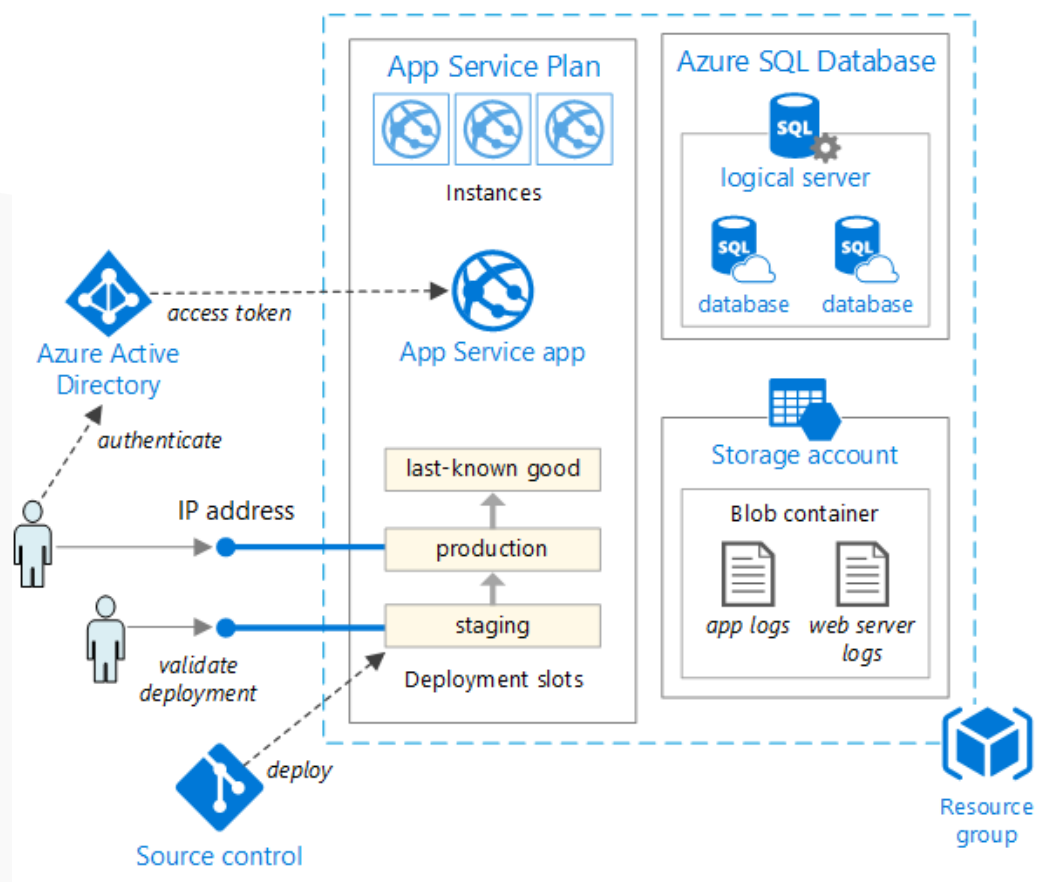
Application Layers

User Interface

Business Logic

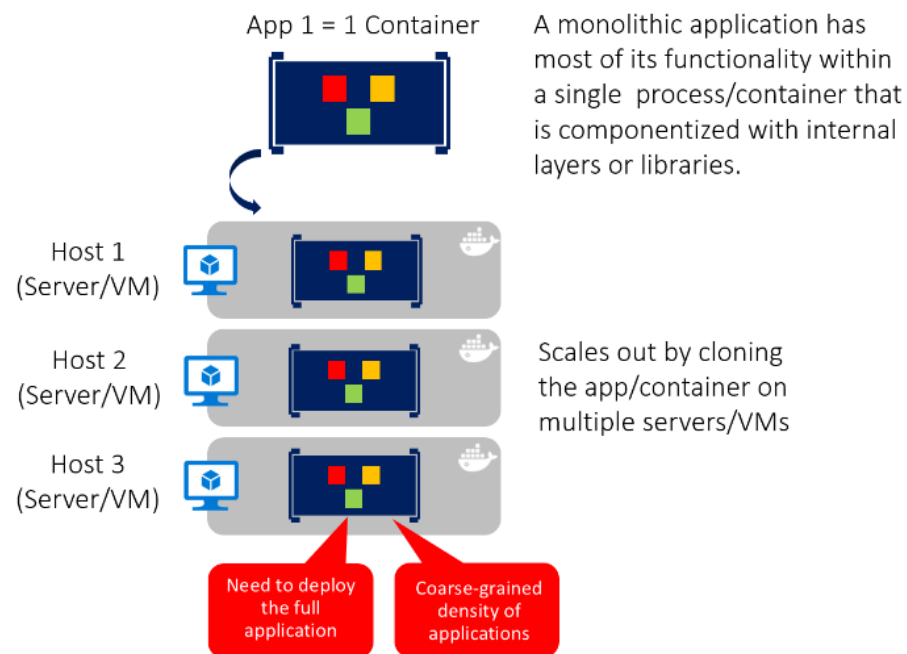
Data Access

# Arquitetura em camadas (Tiers)



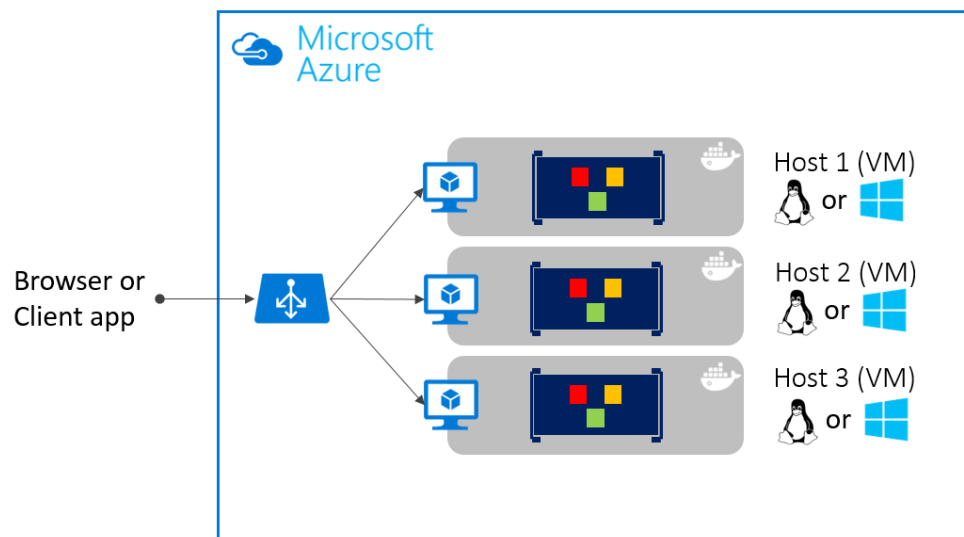
# Arquitetura Monolítica

## Monolithic Containerized application



# Arquitetura Monolítica

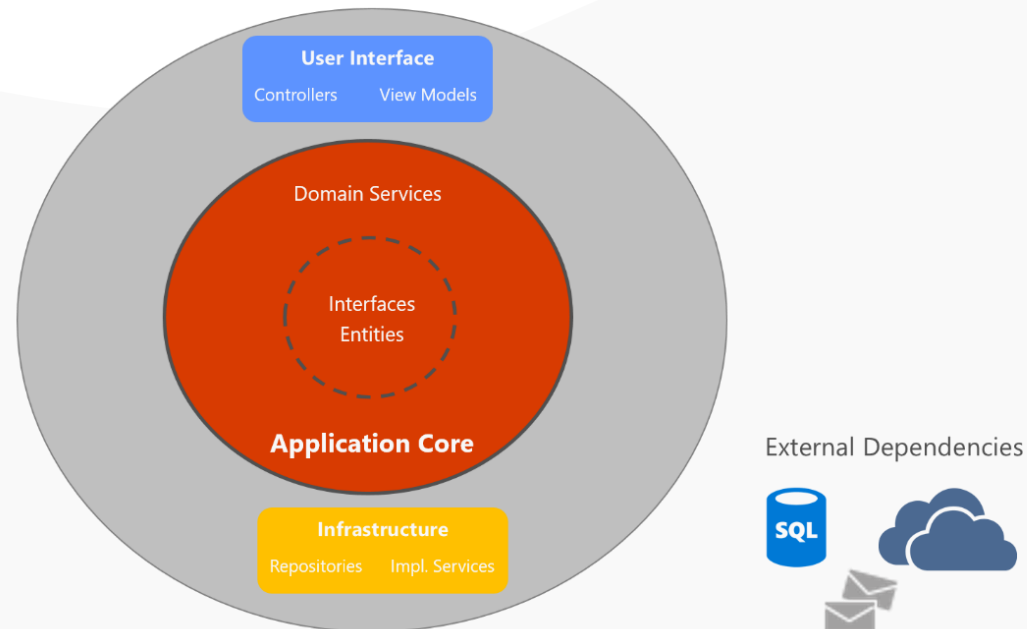
## Architecture in Docker infrastructure for monolithic applications



# Clean Architecture



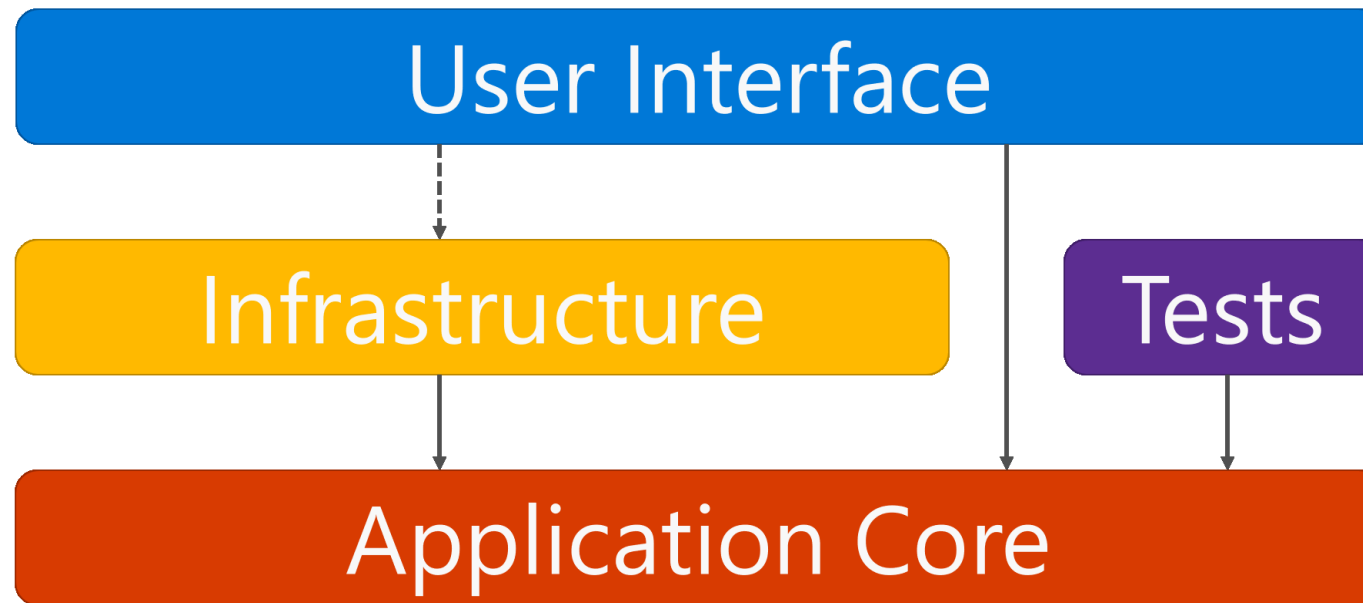
## Clean Architecture Layers (Onion view)



# Clean Architecture

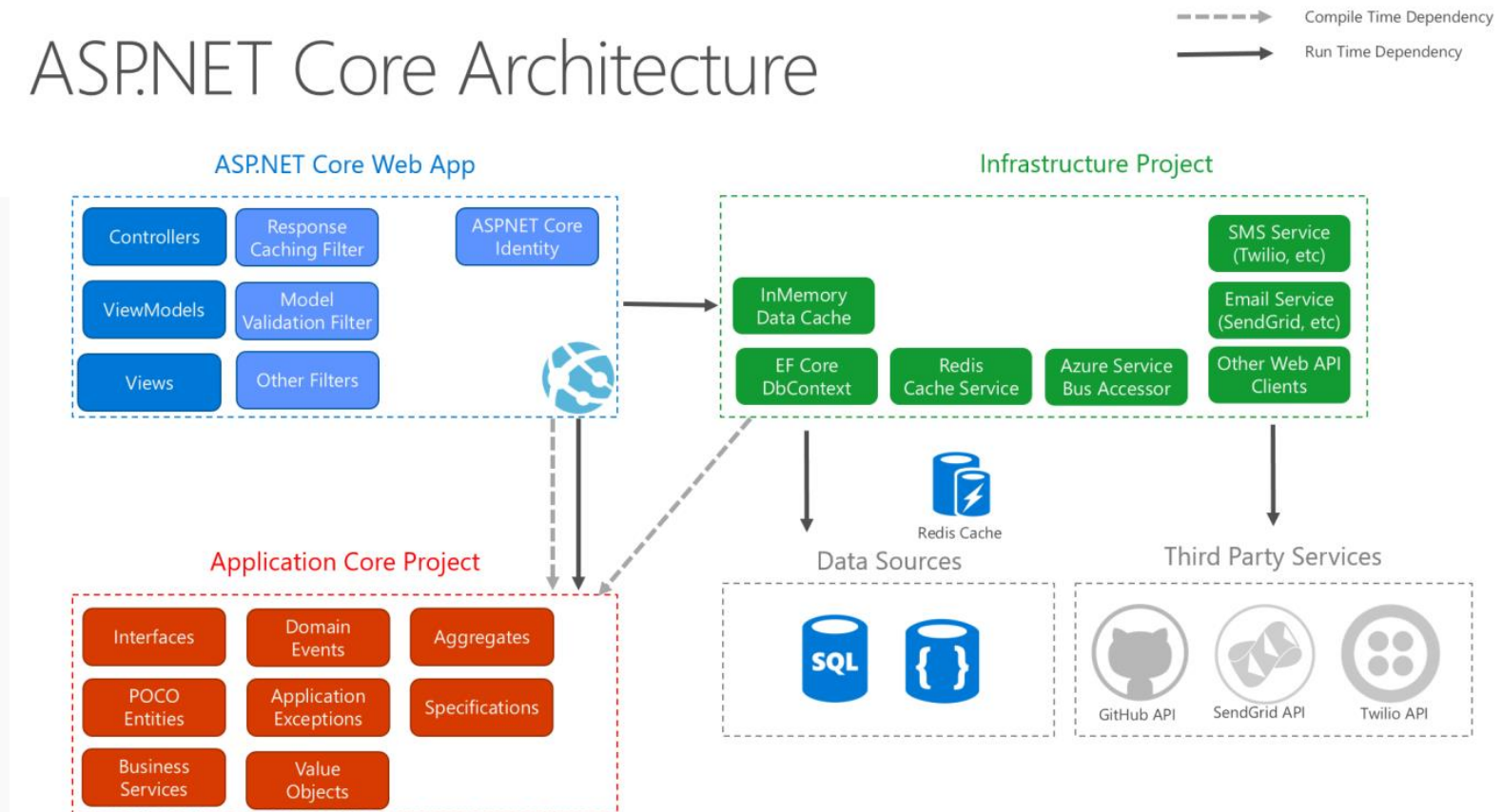
## Clean Architecture Layers

-----> Optional Compile-Time Dependency  
—————> Compile-Time Dependency



# Clean Architecture

## ASP.NET Core Architecture

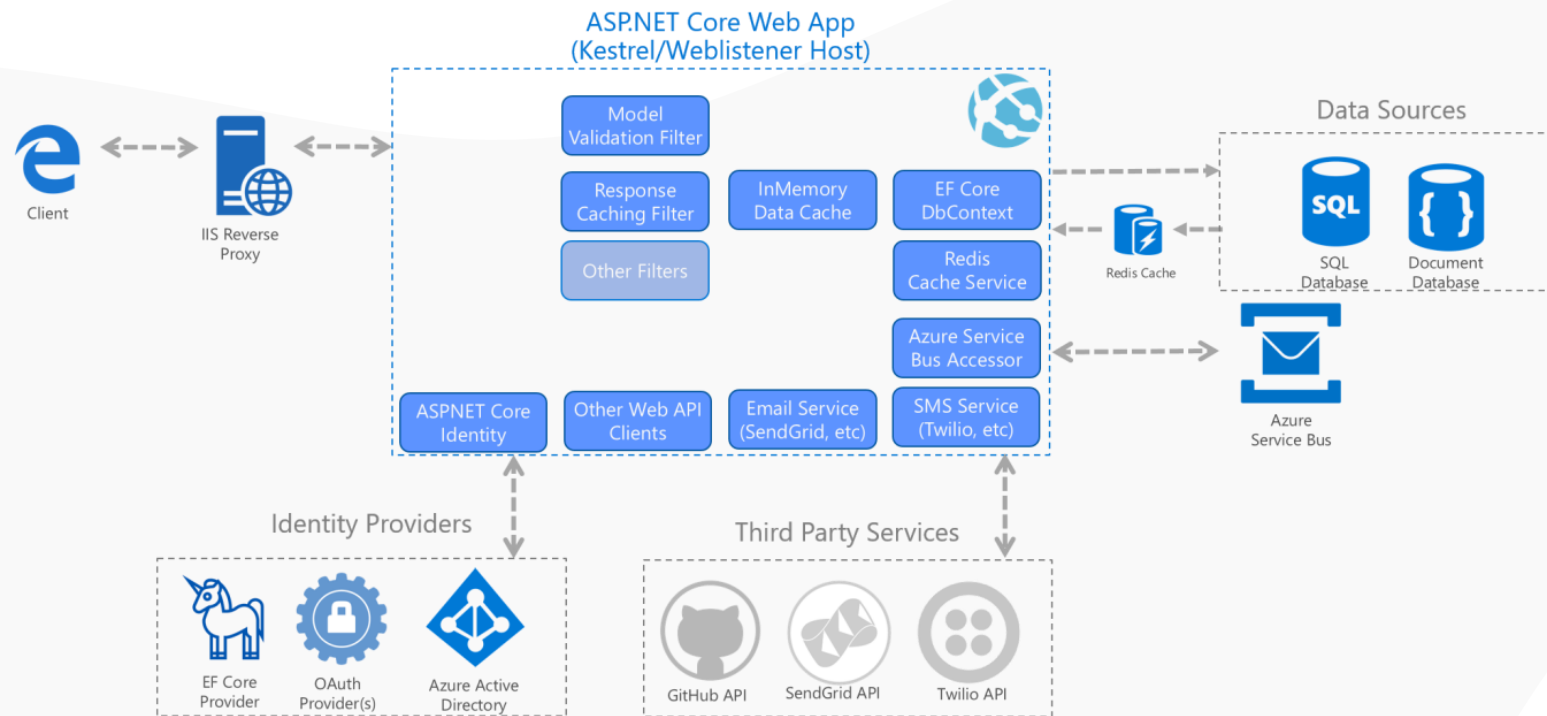




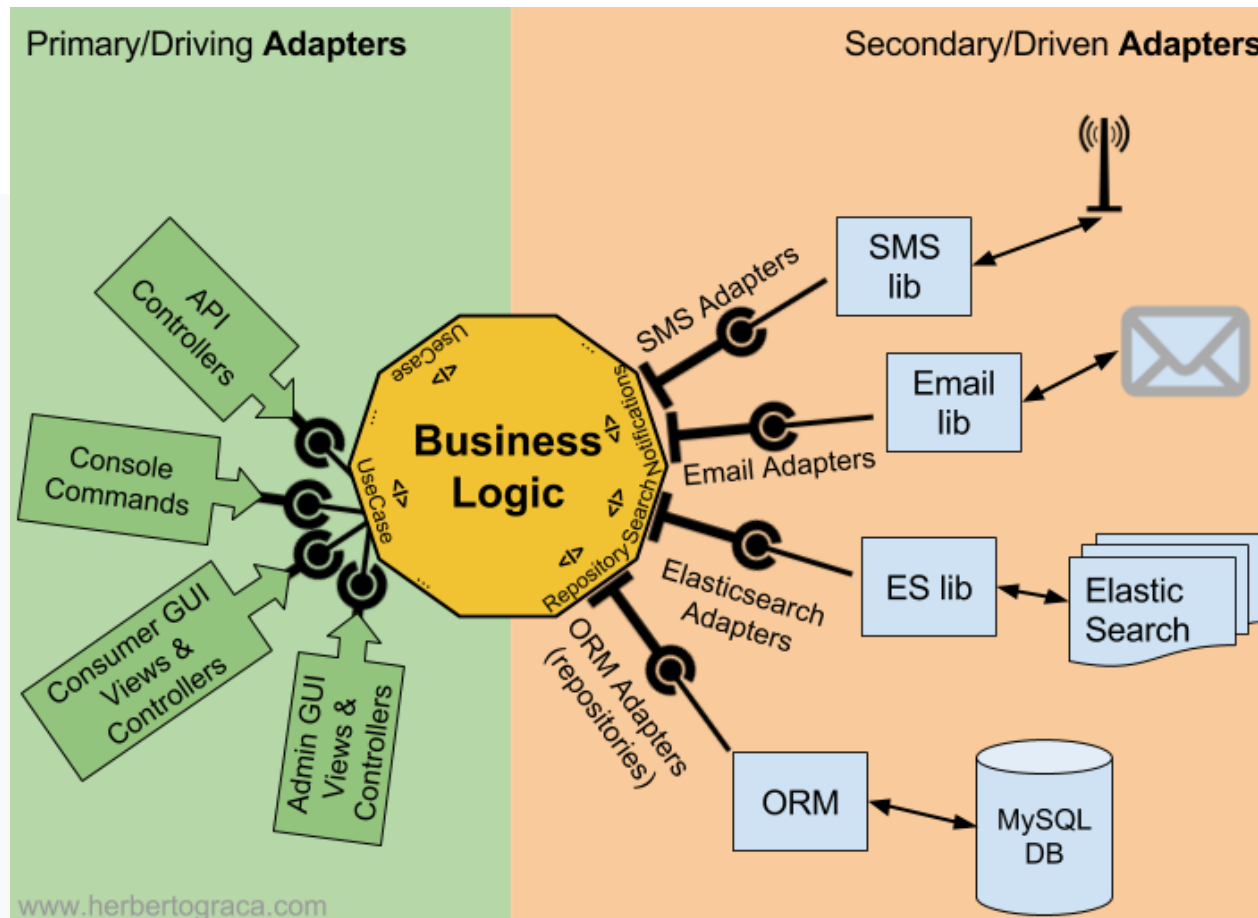
# Clean Architecture



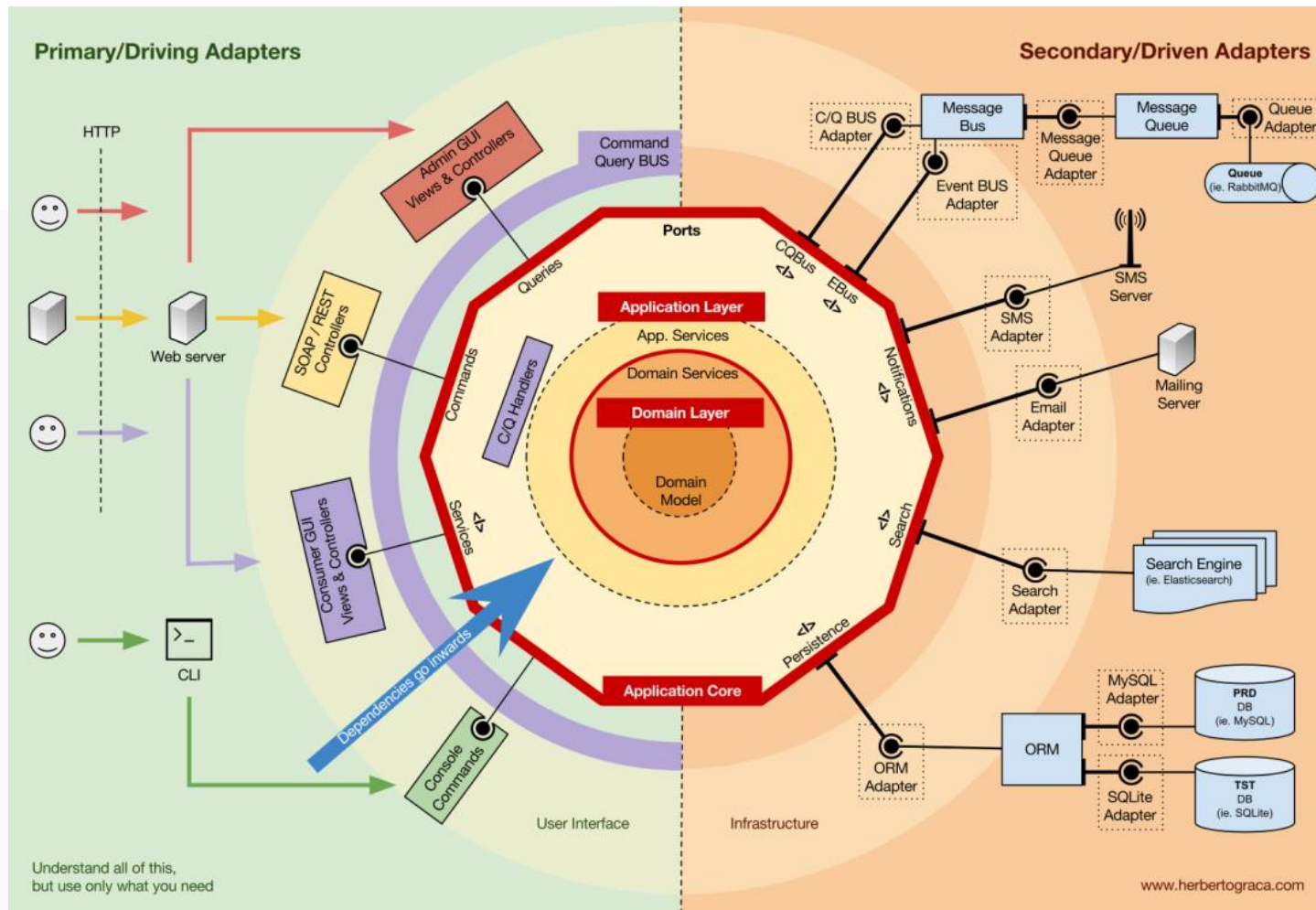
## ASP.NET Core Architecture



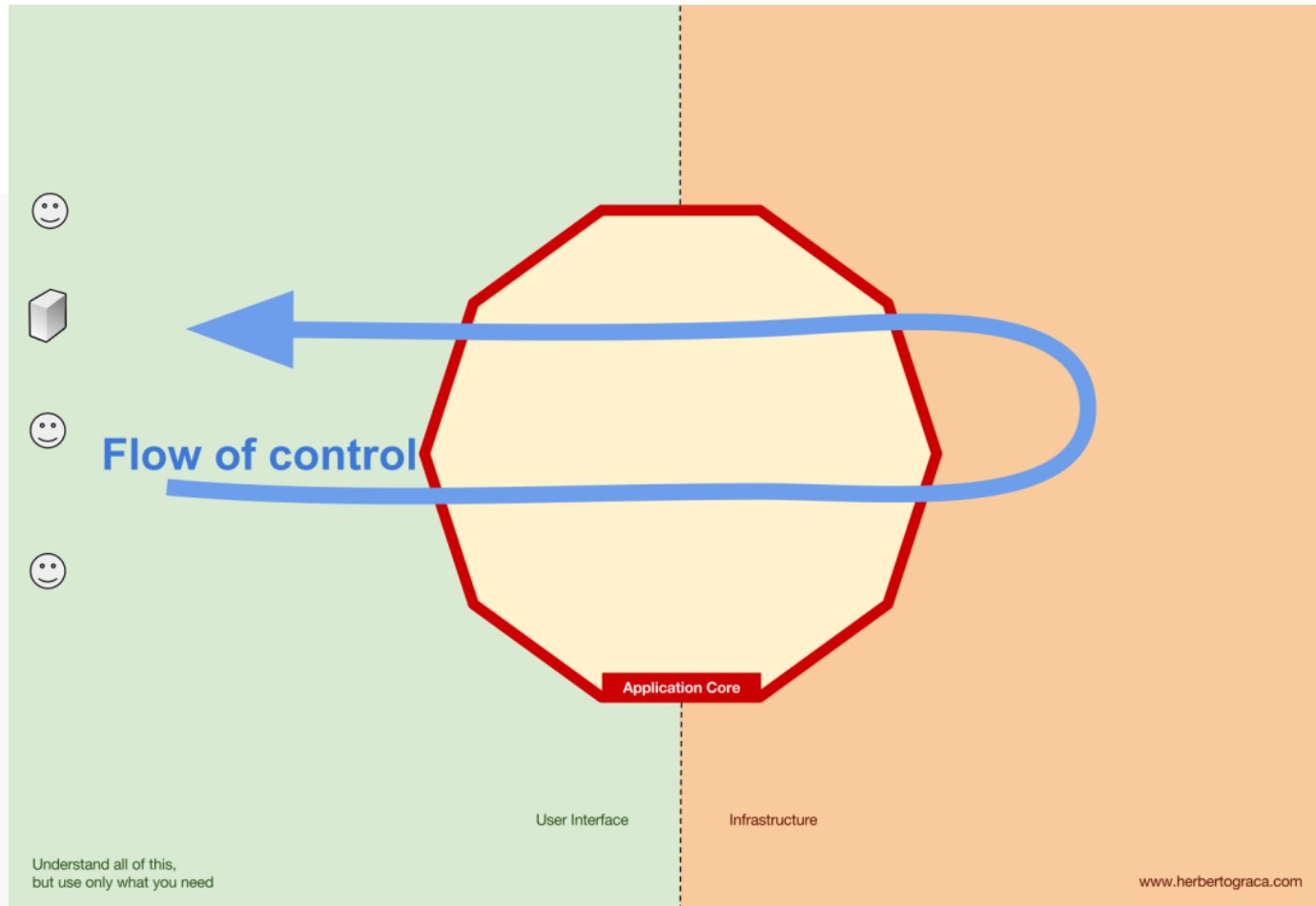
# Hexagonal Architecture



# Hexagonal Architecture



# Hexagonal Architecture



# Arquitetura orientada a eventos



# Arquitetura orientada a eventos





# Arquitetura Orientada a Eventos

- **Pub/Sub**
  - a infraestrutura de mensagens acompanha o controle de assinaturas. Quando um evento é publicado, ele envia o evento para cada assinante. Depois que um evento é recebido, ele não pode ser reproduzido e não será exibido para assinantes novos.
- **Streaming de eventos**
  - eventos são gravados em um registro. Os eventos são ordenados e duráveis. Os serviços não assinam o fluxo, em vez disso, um cliente pode ler a partir de qualquer parte do fluxo. O serviço é responsável por avançar a sua posição no fluxo. Isso significa que um serviço pode reagir a qualquer evento que já tenha sido gerado.

# Arquitetura Orientada a Eventos

- **Processamento de eventos simples.**
  - Uma Azure Functions que é executada como uma imagem é inserida em um Blob.
- **Processamento de eventos complexos.**
  - Um consumidor processa uma série de eventos, procurando padrões nos dados de eventos usando uma tecnologia, como o Azure Stream Analytics ou o Apache Storm. Por exemplo, você pode agregar as leituras de um dispositivo por uma janela de tempo e gerar uma notificação se a média móvel ultrapassar um certo limite.
- **Processamento de stream de eventos.**
  - Use uma plataforma de stream de dados, como o Hub IoT do Azure ou o Apache Kafka, como um pipeline para encaminhar eventos aos consumidores. Os stream processors agem para processar ou transformar o fluxo. Podem haver vários stream processors para subsistemas diferentes do aplicativo. Esta abordagem é uma boa opção para cargas de trabalho de IoT.



# Arquitetura Orientada a Eventos

- **Processamento de eventos simples.**
  - Uma Azure Functions que é executada como uma imagem é inserida em um Blob.
- **Processamento de eventos complexos.**
  - Um consumidor processa uma série de eventos, procurando padrões nos dados de eventos usando uma tecnologia, como o Azure Stream Analytics ou o Apache Storm. Por exemplo, você pode agregar as leituras de um dispositivo por uma janela de tempo e gerar uma notificação se a média móvel ultrapassar um certo limite.
- **Processamento de stream de eventos.**
  - Use uma plataforma de stream de dados, como o Hub IoT do Azure ou o Apache Kafka, como um pipeline para encaminhar eventos aos consumidores. Os stream processors agem para processar ou transformar o fluxo. Podem haver vários stream processors para subsistemas diferentes do aplicativo. Esta abordagem é uma boa opção para cargas de trabalho de IoT.

# Conclusão

- O conhecimento de fundamentos é essencial para desenhar boas soluções
- Exercite a revisão de códigos(Code Review) com o time
- No fim, o bom e velho paradigma Orientado a Objetos nos direciona para melhores desenhos
- Divida para conquistar

# Próxima aula



01.

Principais Arquiteturas

02.

Onion Architecture

03.

Clean Architecture

04.

Hexagonal Architecture

# Principais Arquiteturas de Software da Atualidade

CAPÍTULO 2. API

PROF. Albert Tanure

# Principais Arquiteturas de Software da Atualidade

---

Aula 2.1. API, REST

PROF. Albert Tanure

# Nesta aula



- ☐ API
- ☐ REST e RESTFull
- ☐ Recursos
- ☐ Verbos
- ☐ Http Status Code

# API



Application

Interface

# API

Programming

# API

IGTI

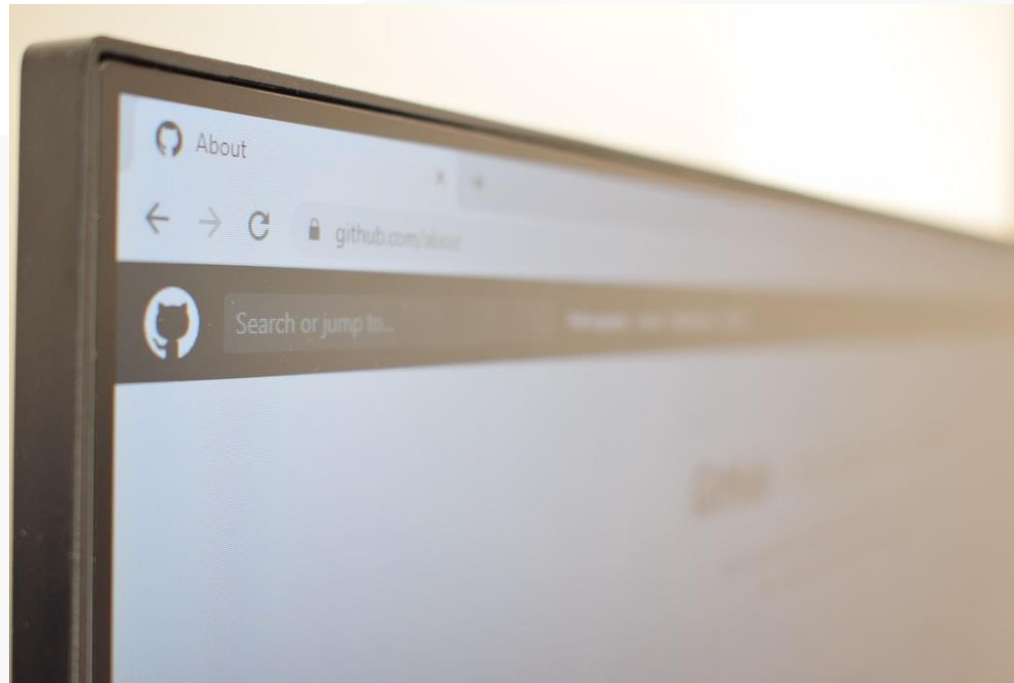




# REST



Http

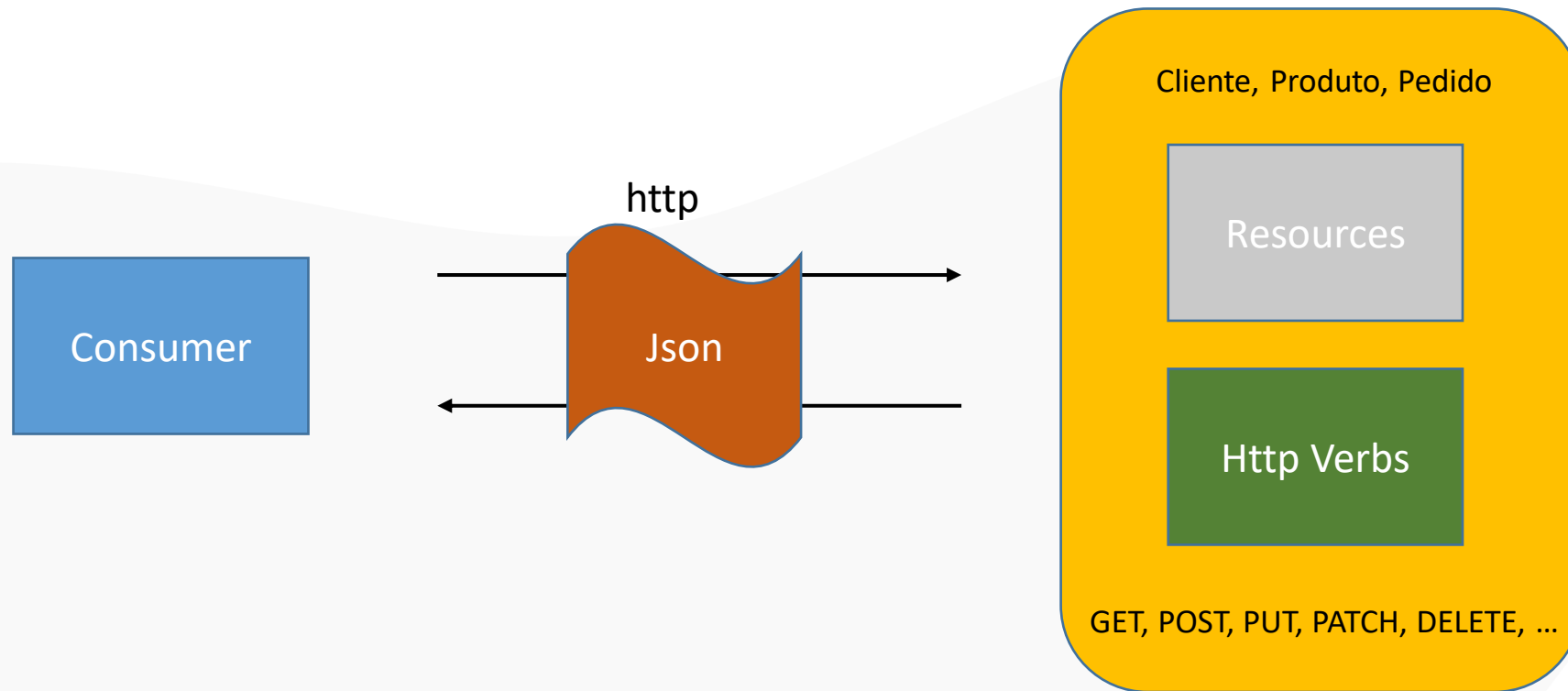


# REST

IGTi



# REST



# REST



| Resource            | POST                              | GET                                 | PUT   | DELETE                           |
|---------------------|-----------------------------------|-------------------------------------|---|----------------------------------|
| /customers          | Create a new customer             | Retrieve all customers              | Bulk update of customers                      | Remove all customers             |
| /customers/1        | Error                             | Retrieve the details for customer 1 | Update the details of customer 1 if it exists | Remove customer 1                |
| /customers/1/orders | Create a new order for customer 1 | Retrieve all orders for customer 1  | Bulk update of orders for customer 1          | Remove all orders for customer 1 |

# REST



- Em conformidade com a semântica HTTP
- Versionamento
- Stateless

# DEMO

# Conclusão

- APIS são mais do que serviços
- REST é um paradigma arquitetural que permite a integração entre aplicações geralmente utilizando HTTP.
- Ao utilizar REST trabalhos com recursos, independente dos consumidores
- Não depende de linguagens específicas

# Próxima aula



01.

Versionamento de API

02.

Boas práticas

03.

Documentação

04.

HATEOAS