Arquiteturas e API's WEB

Laboratório Web

UNIVERSIDADE da MADEIRA

Região Autónoma da Madeira
Governo Regional

startup madeira

Arquitetura Cliente Servidor

- Divide uma aplicação em duas partes, 'cliente' e 'servidor’
- A aplicação é implementada numa rede de computadores, que conecta o cliente ao servidor
- A parte do servidor dessa arquitetura fornece a funcionalidade central: ou seja, qualquer número de clientes pode se conectar ao servidor e solicitar a execução de uma tarefa
- O servidor aceita essas solicitações, executa a tarefa necessária e retorna quaisquer resultados ao cliente, conforme apropriado


HTTP
REQUEST
Browser
Web
Server
RESPONSE
HTTP

HTTP: A SET OF RULES (AND A FORMAT) FOR DATA BEING TRANSFERRED ON THE WEB.

Stands for ‘HyperText Transfer Protocol’. It’s a format (of various) defining data being transferred via TCP/IP.

Métodos HTTP

- **GET** - This is used to provide a read only access to a resource
- **POST** - This is used to create a new resource
- **DELETE** - This is used to remove a resource
- **PUT** - This is used to update an existing resource or create a new resource

HTTP REQUEST

```
CONNECT www.google.com:443 HTTP/1.1
Host: www.google.com
Connection: keep-alive
```

HTTP RESPONSE

Status

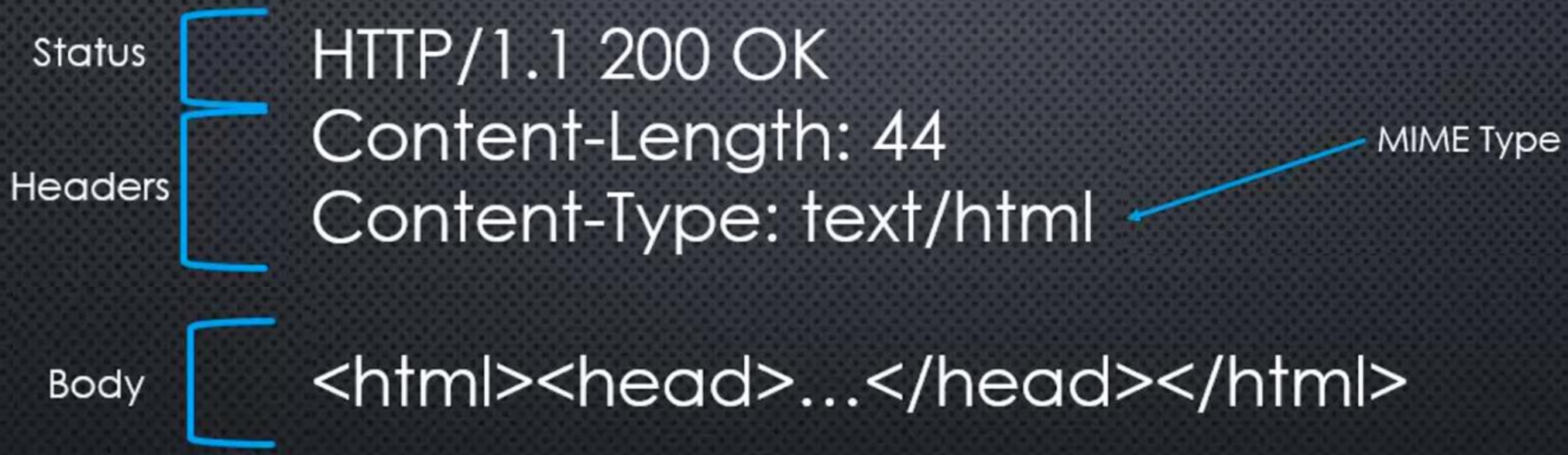
```
HTTP/1.1 200 OK
Content-Length: 44
Content-Type: text/html

<html><head>...</head></html>
```

HTTP RESPONSE

HTTP/1.1 200 OK
Content-Length: 44
Content-Type: text/html

<html><head>...</head></html>

HTTP RESPONSE



MIME type: A STANDARD FOR SPECIFYING THE TYPE OF DATA BEING SENT.

Stands for 'Multipurpose Internet Mail Extensions'.

Examples: application/json, text/html, image/jpeg

Arquitetura monolítica

- As aplicações desenvolvidas contém todo o código em uma única base de código, em .NET Core, por exemplo, toda a aplicação estará dentro de um único ficheiro de solução
- Normalmente a aplicação apenas se conecta a uma base de dados
- É uma abordagem muito comum, mais fácil de implementar e muito menos complexa do que uma arquitetura de microsserviços
- Se tal aplicação tiver que ser escalada horizontalmente, a aplicação inteira é duplicada em vários servidores ou máquinas virtuais

Arquitetura monolítica

- UIL – A camada da interface gráfica apresenta os dados e as funcionalidades ao utilizador

- BLL – Camada que coordena a aplicação, executa comandos, decisões, cálculos e movimenta os dados entre as camadas adjacentes

- DAL – Camada que fornece acesso simplificado a dados armazenados em armazenamento persistente

Arquitetura monolítica - Vantagens

- **Simples de desenvolver** - os IDEs e outras ferramentas focam-se na construção de uma única aplicação
- **Fácil de fazer mudanças radicais na aplicação**— podemos alterar o código, a base de dados, compilar e efetuar o *deployment*
- **Simples de testar** – facilidade de implementar testes unitários, *end-to-end,* invocar a API REST e testar a UI
- ***Deployment*** – apenas temos que efetuar o deployment de uma única aplicação
- **Fácil de escalar** – podemos criar novas instâncias da aplicação consoante a necessidade

Arquitetura monolítica - Desvantagens

Manutenção difícil- Inicialmente, é fácil de manter, mas com o tempo a aplicação cresce e dificulta a gestão. É mais adequado para aplicações de pequena ou média dimensão

Disponibilidade- Quando são efetuadas alterações e é necessário efetuar um deployment, toda a aplicação ficará indisponível durante esse processo

Dificil gestão – Em equipas numerosas trabalhando no mesmo projeto, a probabilidade de conflitos ocorrerem quando o código for *merged* no repositório aumenta, uma alteração feita por uma equipa pode afetar algo que outra equipa está a trabalhar

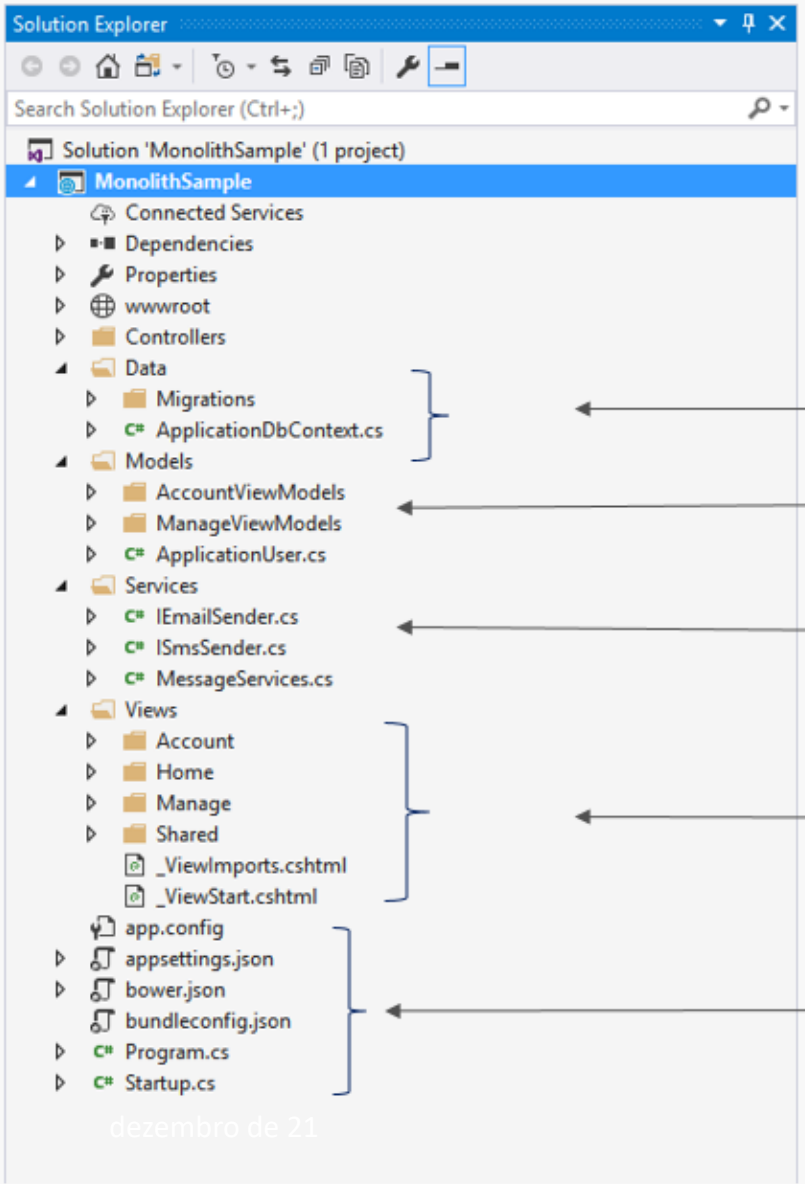
Escalabilidade- Só é possível escalar a aplicação completa. Se a aplicação receber muitas solicitações em apenas uma parte específica da aplicação, não podemos dimensionar apenas esta parte, será necessário dimensionar toda a aplicação

Restrição das tecnologias - Geralmente, quando uma aplicação monólitica é criada, ela provavelmente usará a mesma tecnologia por vários anos

Quando devemos utilizar esta arquitetura?

- Quando a aplicação a implementar for de pequena a média dimensão
- Quando o projeto for uma prova de conceito, permite uma rápida iteração
- Quando a equipa é pequena (2 a 5 membros)
- Quando a escala e a complexidade necessária é previsível

VS Solution Structure



Data Access Logic
- EF Migrations
- EF DbContext and model design

UI Models

Application Services (interfaces and implementations)

Presentation Logic

Application Entry Point and Configuration

Arquitetura por camadas

- Dividir a aplicação por responsabilidades e contextos

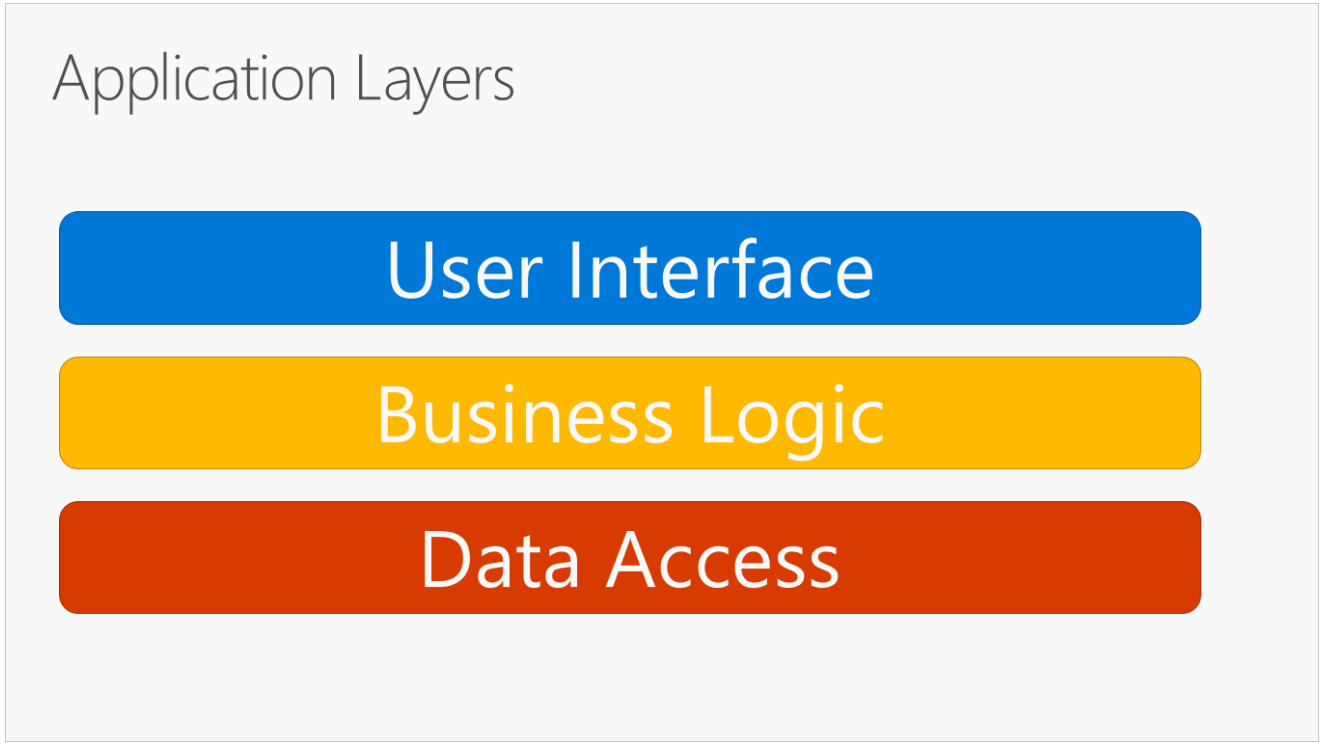
- Ao organizar o código em camadas, a funcionalidade comum de baixo nível pode ser reutilizada em toda a aplicação. Essa reutilização é benéfica porque significa menos código que é escrito reforçando o princípio de não se repita (DRY)

- Podemos impor restrições sobre quais camadas podem se comunicar com outras camadas. Quando uma camada é alterada ou substituída, apenas as camadas que interagem com ela devem ser afetadas. Limitando quais camadas dependem de outras, o impacto das alterações pode ser mitigado para que uma alteração não afete toda a aplicação

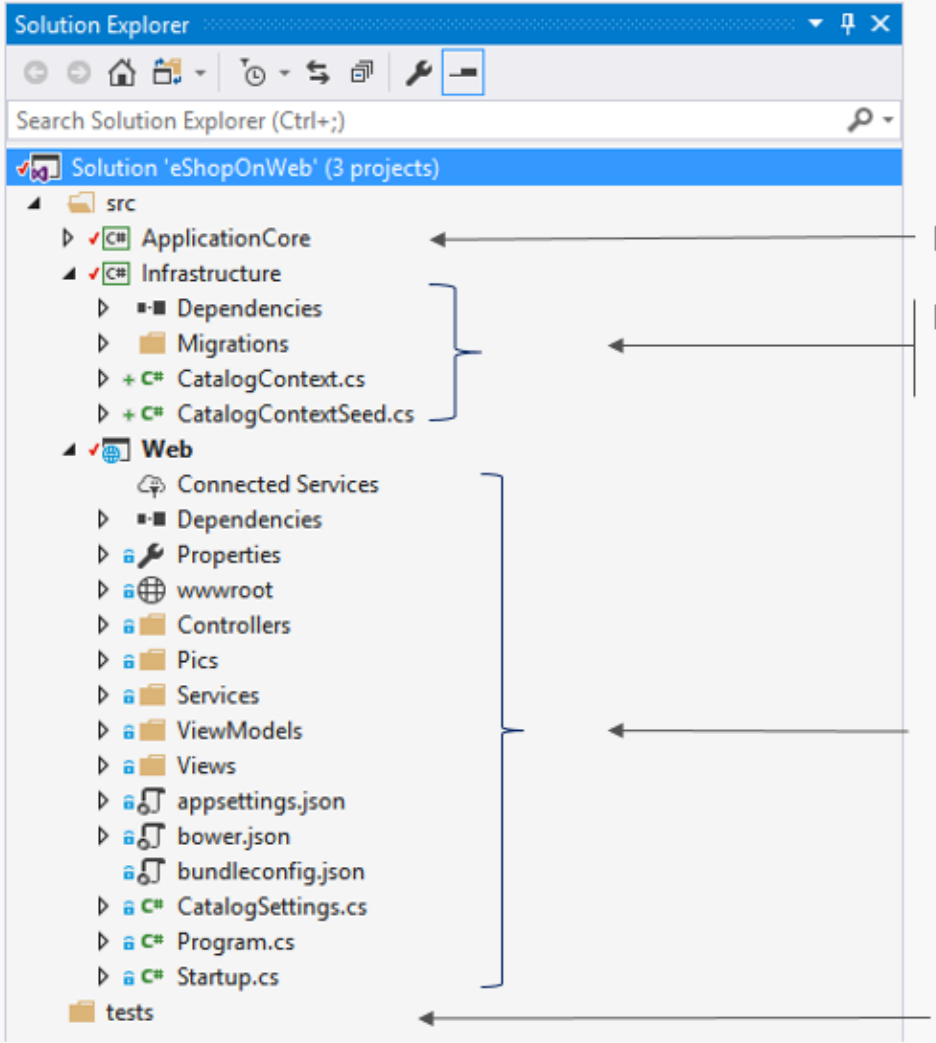
- Camadas (e encapsulamento) tornam muito mais fácil substituir funcionalidades dentro da aplicação. Por exemplo, podemos inicialmente usar a nossa própria base de dados, mas posteriormente podemos escolher usar uma BD baseada em cloud ou uma API web

Arquitetura por camadas

- Os utilizadores efetuam pedidos através da camada de UI, que interage apenas com a BLL
- A BLL invoca a DAL para solicitações de acesso a dados
- A camada de UI não deve fazer nenhuma solicitação ao DAL diretamente, nem deve interagir com a persistência de dados
- O BLL só deve interagir com a persistência passando pelo DAL. Dessa forma, cada camada tem a sua própria responsabilidade bem definida



VS Solution Structure



Business/Application Model

Data Access Logic (Infrastructure)

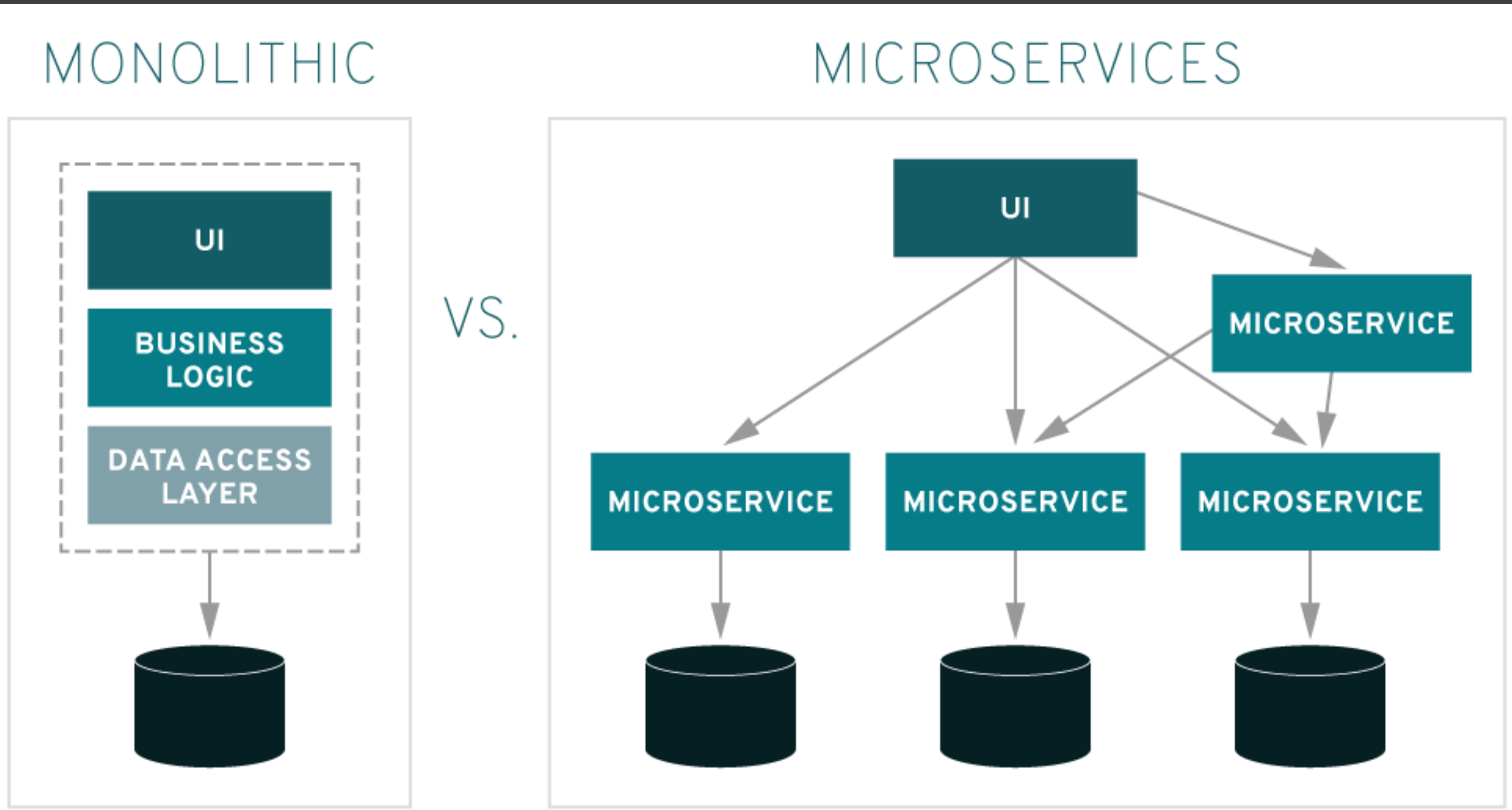
- EF Migrations
- EF DbContext and model design

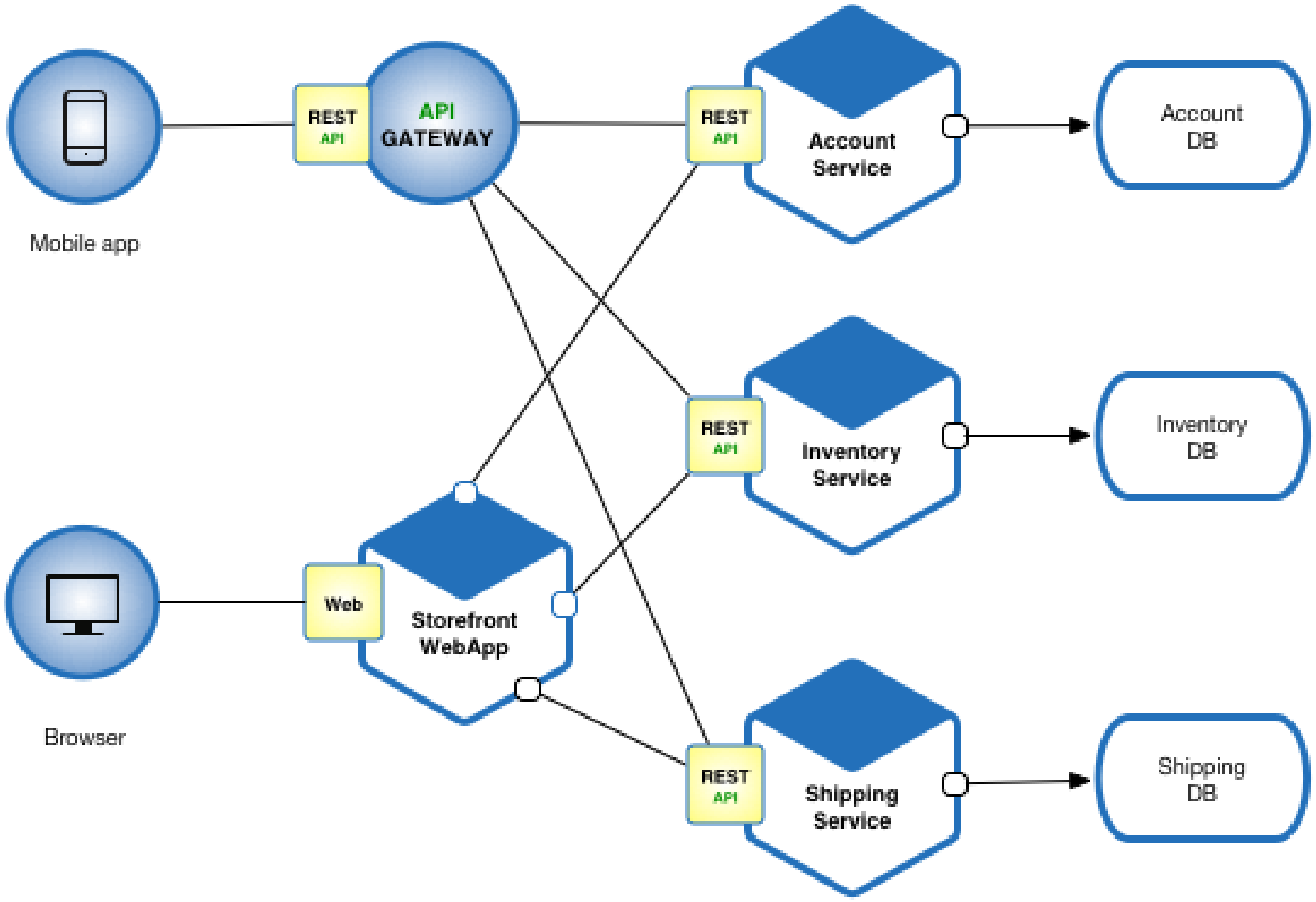
Presentation Logic

Automated Tests

Microsserviços (Microservices)

- Aqui a aplicação é decomposta em funções básicas
- Cada função é denominada um serviço e pode ser criada e *deployed* de maneira independente
- Isso significa que cada serviço individual pode funcionar ou falhar sem comprometer os restantes
- Os microsserviços comunicam entre si através de interfaces de programação de aplicações (APIs) independentes da linguagem de programação

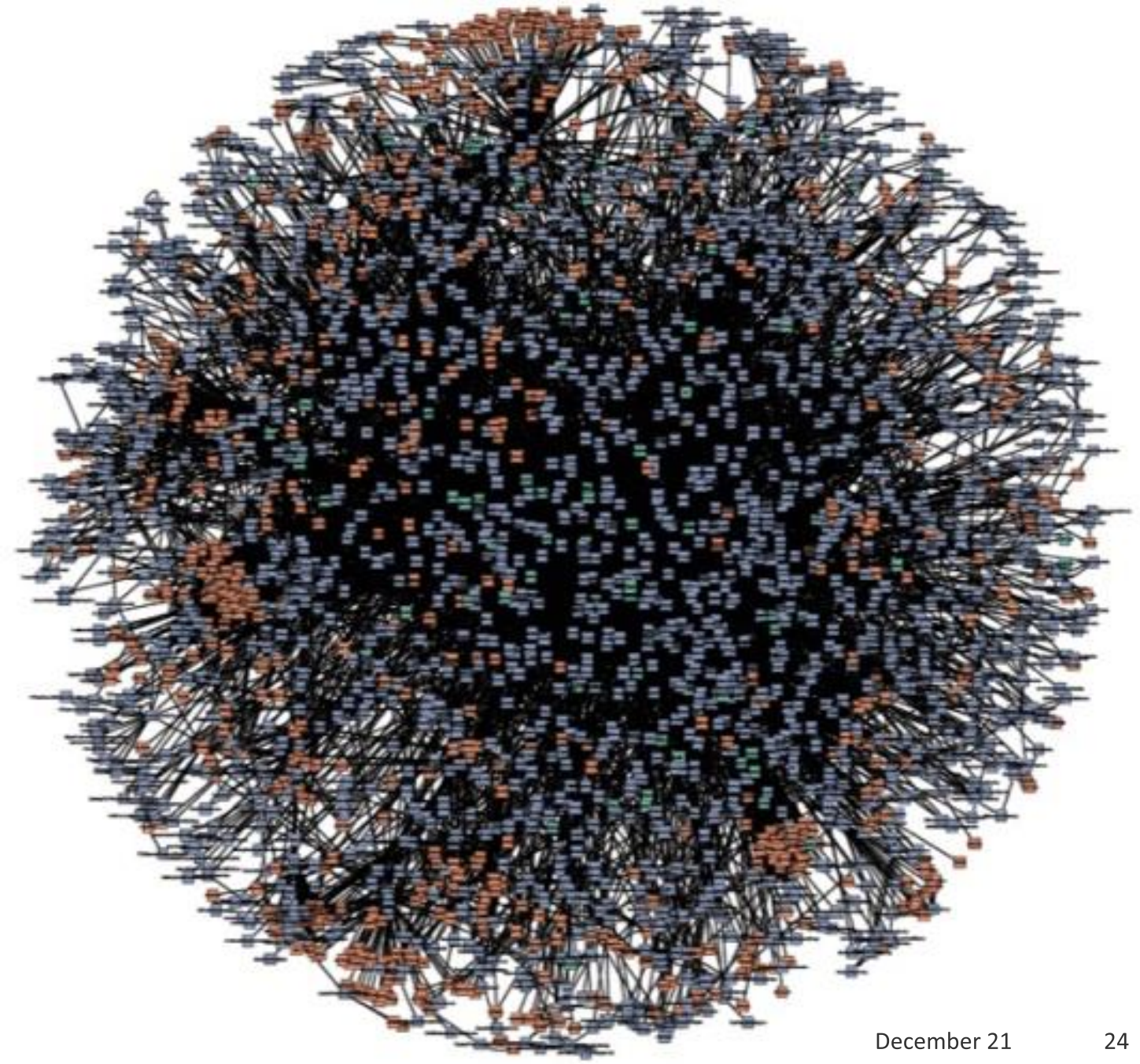

MONOLITHIC
UI
BUSINESS LOGIC
DATA ACCESS LAYER
VS.
MICROSERVICES
UI
MICROSERVICE
MICROSERVICE
MICROSERVICE
MICROSERVICE

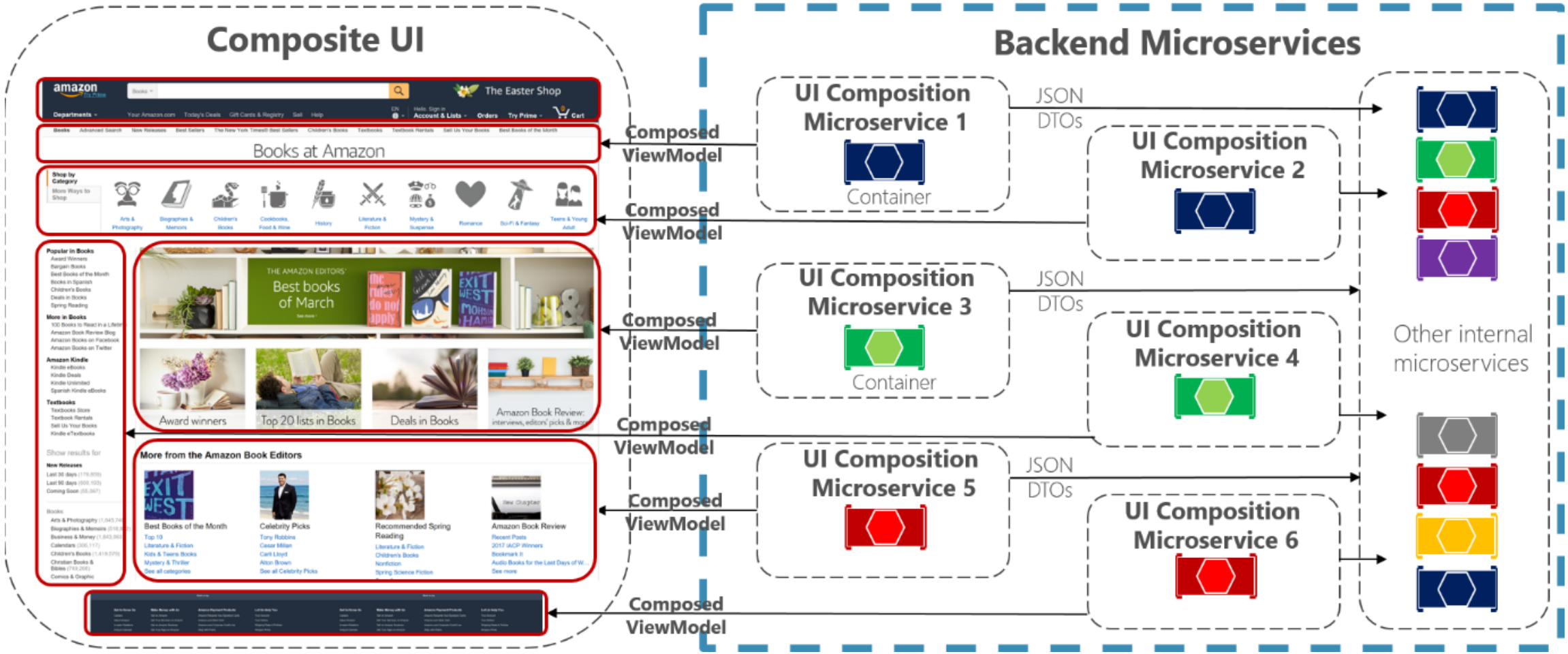

Mobile app
REST
API
API
GATEWAY
REST
API
Account
Service
Account
DB
REST
API
Inventory
Service
Inventory
DB
Browser
Web
Storefront
WebApp
REST
API
Shipping
Service
Shipping
DB

Microsserviços (Microservices)

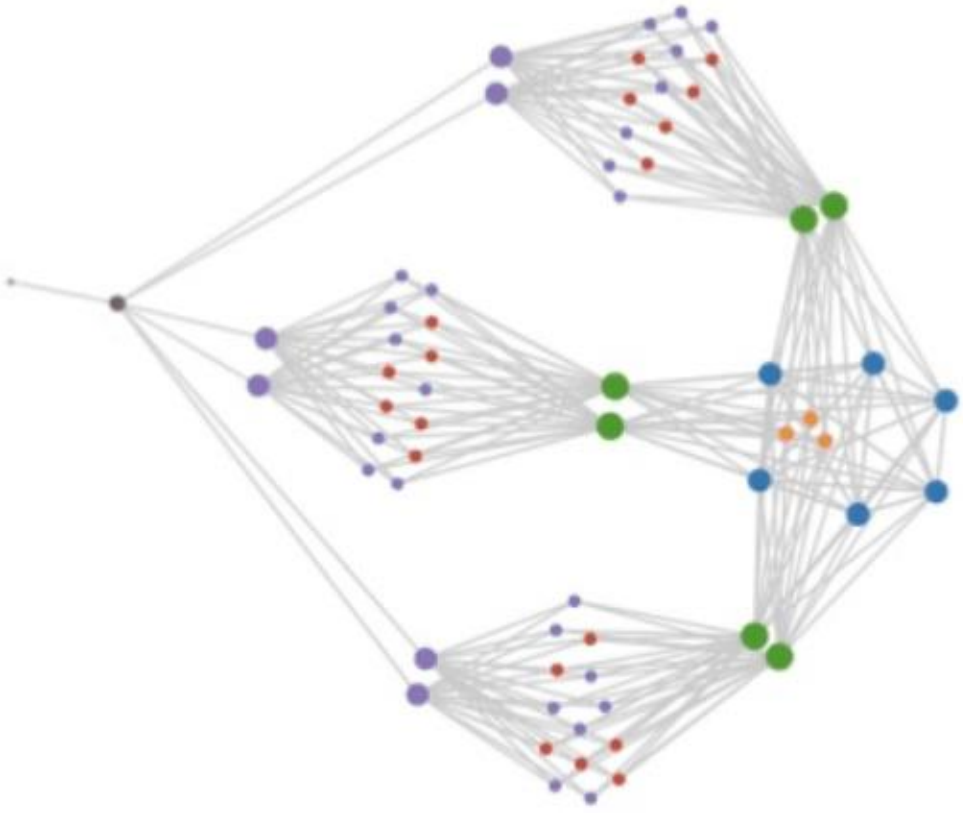
- Cada microsserviço pode ter a sua base de dados
- Os serviços são acoplados de forma fraca (loose coupling), significa que os serviços podem mudar sem impactar os clientes (quem consome o serviço)
- É uma arquitetura poliglota, já que cada serviço pode ser implementado com a melhor linguagem e tecnologia para a função necessária
- Atualmente são os pilares das aplicações nativas em cloud

Arquitetura Amazon




Composite UI
Backend Microservices
Books at Amazon
Best books of March
Award winners
Top 20 lists in Books
Deals in Books
More from the Amazon Book Editors
Best Books of the Month
Celebrity Picks
Recommended Spring Reading
Amazon Book Review
Composed ViewModel
Composed ViewModel
Composed ViewModel
Composed ViewModel
Composed ViewModel
UI Composition Microservice 1
Container
UI Composition Microservice 2
UI Composition Microservice 3
Container
UI Composition Microservice 4
UI Composition Microservice 5
UI Composition Microservice 6
JSON DTOs
JSON DTOs
JSON DTOs
Other internal microservices

Arquitetura Netflix



Simplified Architecture

Actual Architecture

Microsserviços - Vantagens

- Ciclos de desenvolvimento mais curtos, compatibilidade com atualizações e *deployments* mais ágeis
- Altamente escalável, já que é possível aumentar as instâncias dos serviços que estão a ter mais procura (e vice-versa)
- Os serviços se construídos corretamente não afetam os demais. Se um falhar, o restante da aplicação permanece em funcionamento (Exemplos?)
- Os serviços são mais modulares e menos complexos do que as aplicações monolíticas tradicionais

Microsserviços - Desvantagens

- Complexidade maior para gerir todos os serviços da aplicação, também implica algum desperdício de recursos computacionais
- Mais “peças” onde é necessário efetuar monitorização e acompanhamento
- Comunicação sujeita a falhas, existe a necessidade de sincronização dos dados entre os serviços, normalmente é usado um sistema de mensagens (message broker)
- Testes entre serviços podem ser extremamente complexos

Quando utilizar Microsserviços?

- Se a aplicação a desenvolver, for altamente requisitada mas apenas em determinados momentos (IRS, Black Friday)

- Diversas equipas com pilhas tecnológicas distintas, cada equipa pode trabalhar num microsserviço diferente

- Garantir resiliência e tolerância a falhas, mesmo que algum dos seus componentes falhe, essa indisponibilidade não deverá afetar o funcionamento dos restantes

Referências

Published by:

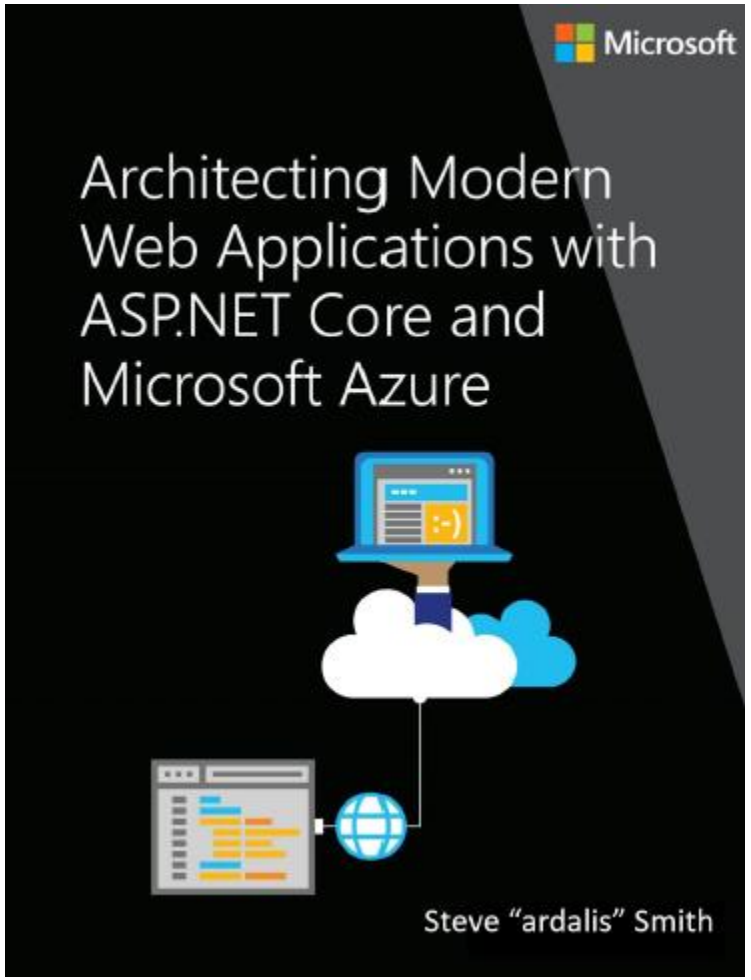
Manning Publications

Topics:

Design Patterns

Microservices Patterns

By Chris Richardson




qualificar
TAL
A <PROGRAMAÇÃ