*Uma imagem com texto

Descrição gerada automaticamente*

Mestrado Informática aplicada

Arquitetura de Software

Prof. Fernando Bento

Wilson Cristiano Oliveira

Nº180100286

180100286@esg.ipsantarem.pt

Vasile Timotin

Nº 160173156

160173015@esg.ipsantarem.pt

Escola Superior de Gestão e Tecnologia de Santarém

Complexo Andaluz, apartado 295, 2001-904 Santarém, Portugal

# Microserviços

A arquitetura de microserviços é uma arquitetura de software que organiza uma aplicação como uma coleção de serviços pequenos e autônomos, cada um executando um processo de negócio específico e comunicando-se entre si através de APIs bem definidas.

Esta abordagem contrasta com a arquitetura monolítica tradicional, onde uma aplicação é construída como uma única unidade indivisível.

### Características Principais dos Microserviços:

1. **Autonomia**: Cada microserviço é independente e pode ser desenvolvido, implantado e escalado de forma separada dos outros. Isso facilita a manutenção e a atualização dos serviços sem impactar a aplicação como um todo.
2. **Descentralização**: Em vez de uma base de dados centralizada, cada microserviço pode ter seu própria base de dados, adequado às suas necessidades. Isso permite a escolha das tecnologias mais apropriadas para cada serviço.
3. **Comunicação via APIs**: Os microserviços se comunicam principalmente através de APIs HTTP/REST, mensageria ou outras formas de RPC (Remote Procedure Call). Essa comunicação é geralmente baseada em protocolos leves e padronizados.
4. **Implementação Independente**: Equipas diferentes podem desenvolver e implantar microserviços diferentes, permitindo que múltiplos processos de desenvolvimento ocorram em paralelo.
5. **Escalabilidade**: Os microserviços podem ser escalados de maneira independente. Se um serviço específico precisa de mais recursos, ele pode ser escalado sem a necessidade de escalar a aplicação inteira.

### Benefícios:

* **Flexibilidade**: Permite a adoção de diferentes tecnologias e linguagens de programação, conforme necessário para cada serviço.
* **Resiliência**: Falhas em um serviço específico não necessariamente afetam toda a aplicação, aumentando a robustez.
* **Manutenibilidade**: Facilita a identificação e correção de problemas, além de tornar o código mais modular e escalável.

### Desafios:

* **Complexidade**: A gestão de múltiplos serviços independentes pode ser complexa, exigindo soluções robustas para orquestração, monitoramento e deployment.
* **Comunicação Inter-Serviços**: A comunicação entre serviços pode introduzir latência e falhas de rede, requerendo estratégias para garantir a consistência e a disponibilidade.
* **Teste e Debugging**: Testar e depurar uma aplicação distribuída é mais desafiador do que em uma aplicação monolítica.

### Arquitetura do nosso projeto

A arquitetura dos microserviços apresentada está organizada da seguinte forma, com cada microserviço:

1. **Artigo**
   * Responsável pela gestão dos artigos disponíveis no sistema.
   * Fornece serviços como criação, leitura, atualização e exclusão de artigos.
2. **Encomenda**
   * Gere todo o ciclo de vida das encomendas.
   * Inclui funcionalidades para criação de encomendas, atualização de status, rastreamento e cancelamento.
3. **FrontEnd**
   * Representa a interface de usuário do sistema.
   * Implementa a lógica de apresentação e interações do usuário com o sistema.
4. **Instance**
   * Localização da instância da base de dados.
   * Poderia ser utilizado para criar um monitoramento da aplicação.
5. **PaymentGateway**
   * Responsável pela integração com gateways de pagamento.
   * Gere transações financeiras, validação de pagamentos e retorno de status de pagamento.
6. **Utilizador**
   * Gere os dados e a autenticação dos utilizadores.
   * Fornece serviços de registo, login, atualização de perfil e gerenciamento de permissões.

Uma imagem com diagrama, captura de ecrã, Desenho técnico, Esquema

Descrição gerada automaticamente

### Representação das camadas de Microserviço:

Os nossos microserviços são compostos por um ficheiro app.py que é a Main da aplicação, ela é responsável por instanciar todo o microserviço utilizando um Model.

Temos uma pasta para a Base de Dados, utilizando a bibliotecal SQLAlchemy juntamente com o Flask para disponibilizar os servidores em ambiente desenvolvimento Local.

Os serviços tem um config.py que serve de ficheiro de configuração do microserviço, de forma a melhorar a leitura e manutenção do código, contudo não foi muito aprofundada, e poderá ser melhorada.

Esse Modelo é a definição do contrato da aplicação, que por sua vez tem o acesso a BD, também como a definição da classe e dos seus métodos.

Routes.py é onde temos a definição de todas as rotas do microserviço que é a definição das nossas API REST, de forma a disponibilizar o acesso a informação da Base de dados aos nossos clientes autorizados.

A melhoria criada foi um método de healthcheck que serve para validar se o ambiente está ok e se todos os seus serviços estão no ar, de forma que este serviço é implementado na aplicação de forma de uma classe.

Por sua vez outra melhoria introduzida foi a documentação das API’s através de swagger, que disponibiliza o JSON automaticamente com base no modelo e suas definições nas rotas.

Temos também um microserviço que faz de proxy para os serviços ou sistemas externos de pagamentos, que foi criado via Webservice Outsystems Application.

Todos os microserviços também contem um readme.md para descrever cada microserviço e como deverá ele ser instanciado e executado.

1. **Artigo**

Estrutura de pasta do microserviço:

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

App.py:

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Config.py:

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Health.py:

Uma imagem com texto, captura de ecrã, ecrã, software

Descrição gerada automaticamente

Models.py

Uma imagem com texto, captura de ecrã, software, Sistema operativo

Descrição gerada automaticamente

Routes.py

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

1. **Encomenda**

Estrutura de pastas

Uma imagem com texto, captura de ecrã, Tipo de letra, design

Descrição gerada automaticamente

App.py

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

Config.py

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Health.py

Uma imagem com texto, captura de ecrã, ecrã, software

Descrição gerada automaticamente

Models.py

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Routes.py

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

1. **FrontEnd**

Não vamos investir muito no Front End visto que foi utilizado, mas não era o foco do nosso projeto.

Uma imagem com texto, captura de ecrã, Tipo de letra, design

Descrição gerada automaticamente

1. **PaymentGateway**

**Estrutura do projeot:**

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

App.py

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Config.py

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Health.py

Uma imagem com texto, captura de ecrã, software, ecrã

Descrição gerada automaticamente

Models.py

Esta classe define o modelo da base de dados interna do microserviço que serve de “cacher” dos pagamentos, sendo que é utilizada para guardar os pedidos e atualizar respostas com base na integração externa.

A integração externa é o serviço de Outsystems criado para este âmbito, tentar simular este processo de pagamento, serviço disponível publicamente.

<https://vasile-timotin.outsystemscloud.com/PaymentGateway/rest/Payment/>

Todos os serviços e chamadas para o serviço da payment gateway é feito na função do modelo de forma a integrar os dados na BD e no sistema externo.

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

Routes.py

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

1. **Utilizador**

Estrutura do microserviço:

Uma imagem com texto, captura de ecrã, Tipo de letra, design

Descrição gerada automaticamente

App.py

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

Config.py

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Health.py

Uma imagem com texto, captura de ecrã, software, ecrã

Descrição gerada automaticamente

Models.py

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Routes.py

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, menu

Descrição gerada automaticamente

Documentação automática de cada serviço com o seu swagger, dando liberdade ao utilizador de testar diretamente as API’s no swagger, executando assim o Curl que está por tras.

Uma imagem com texto, captura de ecrã, software, Sistema operativo

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, software, ecrã

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, software, ecrã

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, software, Ícone de computador

Descrição gerada automaticamente

Por sua vez cada swagger tem um swagger.json que facilmente é feito o download de um ficheiro JSON que é importanto em ferramentas como o Postman.

Na imagem abaixo vamos observar um ficheiro e como fica importado no Postman.

Uma imagem com texto, captura de ecrã, ecrã, software

Descrição gerada automaticamente

Importação no Postman:

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Resultado:

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

Utilização do healthcheck do serviço Utilizador.

Uma imagem com captura de ecrã, software, Software de multimédia, texto

Descrição gerada automaticamente

### Conclusão Aplicação do GP:

Podemos então concluir que a arquitetura de microserviços é um modelo de sistemas distribuídos, sendo que a decentralização destes módulos dá vantagens para a extensibilidade do sistema em questão, na sua manutenção e evolução de código.

Contudo há maiores dificuldades de debugging dos fluxos end to end do sistema, sendo que os erros podem ocorrer em sistemas diferentes, sendo que é importantíssimo um sistema de loging e alertas que devem ajudar nesse trabalho.

Toda a comunicação entres os nossos serviços utilizam uma interoperabilidade utilizando integrações REST, sendo utilizado JSON para a comunicação entre serviços ou interfaces API.

Poderíamos utilizar RPC ou outro qualquer mecanismo de comunicação entre os sistemas, sendo que a grande vantagem disto é que cada sistema poderá ser desenhado e desenvolvido com base na sua funcionalidade e com a tecnologia necessária.

No nosso exemplo a aplicação outsystems comunica com os nossos serviços que estão feitos utilizando o flask, também como as Bases de dados são ou podem ser diferentes.

Concluimos também que cada microserviço tem um ciclo de vida próprio, e não dependem uns dos outros para fazerem o seu processamento, contudo se algum serviço parar, poderemos ter impactos em certas funcionalidades do sistema.

### Bibliografia: