Contexto

A componentização desempenha um papel fundamental na arquitetura de um sistemas web modernos, permitindo a construção eficiente, escalável e modular de aplicações online. Ao dividir um sistema em componentes independentes e reutilizáveis, a complexidade do desenvolvimento é reduzida, facilitando a manutenção, atualizações e correções.

Seguindo essa abordagem, cada componente pode ser projetado para cumprir uma tarefa ou entregar um serviço específico, como autenticação de usuários, gerenciamento de banco de dados, interface do usuário, entre outras. Essa modularização facilita o desenvolvimento paralelo por equipes distintas e oferece uma visão clara das interações entre as diferentes partes do sistema.

Para padrões de arquitetura como microsserviços, a componentização é fundamental. Cada microsserviço é uma componente separada que pode ser desenvolvida, implantada e escalada de maneira independente. Para esse contexto, vale reforçar que cada microsserviço deve ser uma unidade autônoma que desempenha uma função específica da aplicação.

É nesse contexto que vamos colocar em prática o conteúdo apresentado ao longo das aulas explorando a ideia de componentes e serviços. **Neste MVP queremos ver você entregando o seu melhor no desenvolvimento de um sistema web em que seus diferentes componentes atuem como serviços autônomos.**

Objetivo

O seu objetivo é desenvolver um sistema composto por no mínimo três módulos que se comunicam, seguindo o padrão *REST* e/ou *GraphQL*. Você não irá desenvolver todos eles, pelo menos um deles deve ser um componente externo (serviço externo como por exemplo o ViaCEP e FakeStore). A persistência de dados deve existir e ser feita utilizando o SQLite, MySQL ou PostgreSQL. Além disso, cada componente desenvolvido deve ter o seu próprio repositório e, na raiz do repositório,

deve existir um Dockerfile com as instruções que possam garantir a sua execução utilizando *containers*.

Cenários

A seguir apresentaremos possíveis cenários de organização de implementação dos módulos. Escolha 1 (um) desses cenários para a construção do seu MVP:

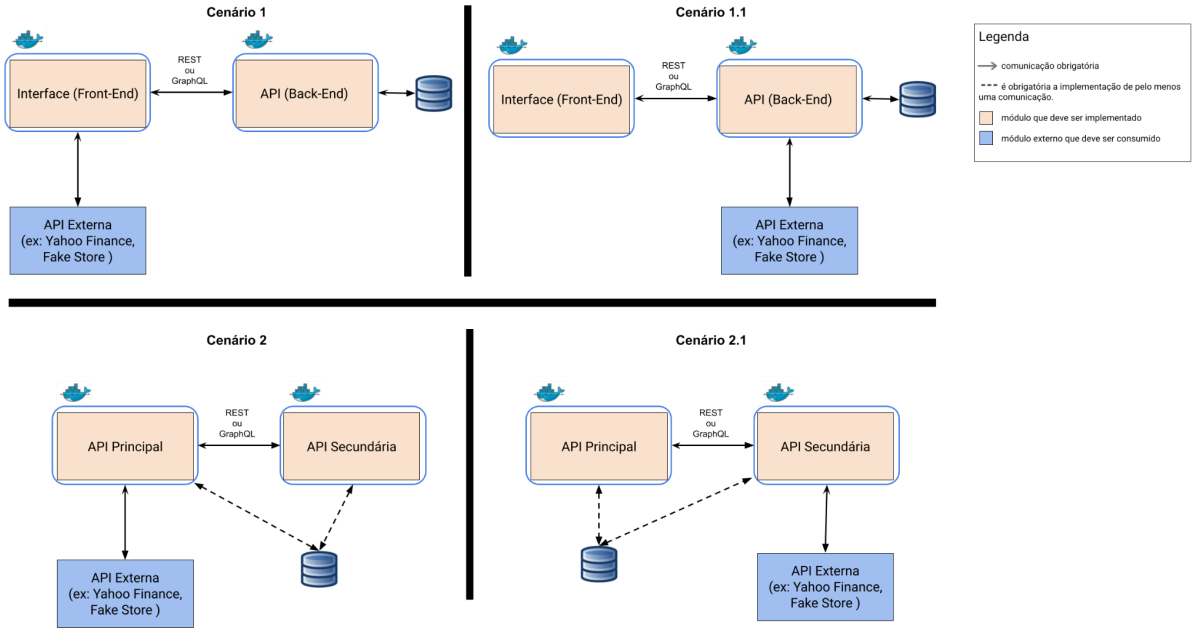

Figura 1. Exemplo de arquiteturas que atendem aos requisitos mínimos.

Cenário 1: A **Interface (Front-End)** pode ser de um sistema de compras online que consulta dados de produtos utilizando um serviço externo (ex: **Fake Store**) e que tem um módulo de compras **API (Back-End)** para efetuar a compra e salvar os registros de informações.

Cenário 1.1: A **Interface (Front-End)** pode ser um *dashboard* financeiro, e o módulo **API (Back-End)** consulta um serviço externo (ex: Yahoo Finances) para trazer dados históricos de ativos financeiros, ou cotação do dólar, etc.

Cenário 2: A **API principal** faz consulta a um serviço externo (ex: ViaCEP) para buscar algum endereço por meio do CEP, e se comunica com uma **API secundária** para calcular a distância entre dois endereços.

Cenário 2.1: A **API principal** pode funcionar como um *proxy* comunicando-se com a **API secundária** que realiza todas as regras de negócios, inclusive se comunicando algum serviço externo.

Antes de Tudo

Antes de começar, certifique-se de instalar o Docker, o Python e todas as demais bibliotecas necessárias em seu computador. Você também precisará de um editor de código de sua preferência, como Visual Studio Code ou outros. Para aqueles que optarem por desenvolver uma *interface* para usuários e se for fazer uso de biblioteca de componentes (React, Next, Vue, etc) certifiquem-se de instalar o Node.js e as dependências necessárias para o desenvolvimento dessa *interface*.

Link para instalação do Docker nos sistemas:

- Windows: https://docs.docker.com/desktop/install/windows-install/
- Ubuntu: https://docs.docker.com/engine/install/ubuntu/
- Mac OS: https://docs.docker.com/desktop/install/mac-install/

Observação: usuários de windows, é importante verificar se a virtualização de sua máquina está ativada na BIOS de sua máquina, pois ela é fundamental para habilitação do WSL2. Em seguida, você deve seguir os passos de instalação e habilitação do WSL2, para execução do Docker.

Requisitos e composição da nota

Pontuação	Requisito
	Para o módulo Interface ou API principal (5,0 pontos)
2,0	**[Caso seja uma API]** Implementar uma API REST em **Python,** como **Flask ou FastAPI**, com pelo menos 4 rotas. Deve existir pelo menos uma para cada um dos métodos: POST, PUT, DELETE, GET.
	Observação : A ausência de um dos métodos implicará na penalização de 0,5 pts.
	[Caso seja uma Interface] Desenvolver uma interface do usuário, utilizando **HTML**, **CSS** e **JavaScript**: ● Será permitido a utilização de bibliotecas ou frameworks baseadas em Javascript, como o React, Next, e outras; ● Será permitido também o uso de bibliotecas de componentes, como o Material UI, Bootstrap, e outras; ● A interface do usuário deve fazer chamadas a pelo menos 4 rotas com métodos HTTP diferentes, isto é: uma GET, uma POST, uma DELETE e uma PUT.
	Observação : A ausência de um dos métodos implicará na penalização de 0,5 pts.
1,0	O README.md deve conter, no mínimo, as seguintes informações: ● Título e uma breve descrição do projeto; ● Instruções de instalação, tais como: descrever as etapas necessárias para que os desenvolvedores possam configurar o ambiente local, instalar dependências, comandos de inicialização, etc. ● Certifique-se de que o arquivo README.md seja formatado de forma clara e use cabeçalhos, listas e formatação de texto para tornar a documentação fácil de ler. ● É obrigatório produzir uma **imagem** (fluxograma) ilustrando a arquitetura da aplicação desenvolvida. Use como exemplo a algum cenário da Figura 1.
1,0	*Dockerfile* para cada componente desenvolvido com todo o processo de implementação da solução em um *container docker*.
	Observação 1: a não execução correta ou não entrega do DockerFile implicará na penalização de 1,0 pts **Observação 2: caso utilize o *docker compose*, o arquivo deve ser disponibilizado na raiz do repositório da interface ou da API principal (-0,5 pts)**
1,0	Criatividade e Inovação: ● Os componentes implementados (na Interface) ou as rotas/modelos de dados (na API) devem ser aplicados em um domínio ou cenário distinto dos exemplos fornecidos. ● API: adicionar funcionalidades extras além do CRUD básico (ex.: autenticação, ordenação, filtros, paginação). ● Interface: propor interações ou visualizações diferentes (ex.: dashboards, carrosséis, feedback visual mais elaborado, gráficos, bibliotecas de componentes, etc).
	Para a API (Back-End) ou API secundária (3,0 pontos)
2,0	Implementação de uma API REST ou GraphQL, com pelo menos 4 rotas.
0,5	O seu README.md deve conter as seguintes informações: ● Título e uma breve descrição do projeto; ● Instruções de Instalação, tais como: descrever as etapas necessárias para que os desenvolvedores possam configurar o ambiente local, instalar dependências, comandos de inicialização, etc. ● Certifique-se de que o arquivo README.md seja formatado de forma clara e use cabeçalhos, listas e formatação de texto para tornar a documentação fácil de ler.

0,5	*Dockerfile* para cada componente desenvolvido com todo o processo de implementação da solução em um *container docker*. **Observação 1: a não execução correta ou não entrega do DockerFile implicará na penalização de 0,5 pts** **Observação 2: caso utilize o *docker compose*, o arquivo deve ser disponibilizado na raiz do repositório da Interface ou da API principal (-0,5 pts)**
Para a API externa (1,0 ponto)	
0,5	Uso de uma **API externa pública** e que ofereça um serviço não pago.
0,5	Apresentar na documentação da Interface ou API principal a API externa que será utilizada, deixando claro informações como: licença de uso (se aplicável), cadastro (se necessário) e rotas que foram utilizados.
	Observação: O consumo da API externa não pode causar o redirecionamento para outra aplicação. Ou seja, você deve consumir e tratar os dados da componente externa na sua aplicação (-1,0 pt).
Organização dos códigos (1,0 ponto)	
1,0	• Cada módulo (Interface e/ou API) deve apresentar um **repositório público separado** no **GitHub**. • A estrutura de pastas e arquivos deve ser organizada de forma clara. • Nomes de arquivos, pastas e componentes devem seguir convenções de boas práticas (ex.: *CamelCase* para componentes React, *snake_case* para arquivos Python).

Atenção! *Caso sejam utilizados como base os exemplos apresentados em aula, serão penalizadas as entregas que não apresentarem pelo menos 50% de código novo.*

Sobre a entrega:

Você deverá produzir um vídeo de **no máximo 6 minutos** que deve seguir o seguinte roteiro: Atenção: caso o tempo seja excedido, haverá desconto de 0,5 ponto na nota do vídeo.

Ordem	Descrição	Sugestão de tempo
1	**Apresentar o objetivo da aplicação desenvolvida.** *- Qual problema você está tentando resolver com a sua aplicação? Qual o propósito dela?*	De 20 a 60 segundos
2	***Apresentação da arquitetura e das estratégias de comunicação implementada***	De 30 a 60 segundos
3	**Apresentação da API externa**	De 30 a 60 segundos
4	**Apresentação da segunda componente implementada.** *- Execução da API (via docker) com você interagindo com todas as rotas implementadas*	De 60 a 90 segundos
5	**Apresentação da componente principal:** *- **[caso seja uma API]** execução da API (via docker) com você interagindo com todas as rotas implementadas no Swagger.* *- **[caso seja um Front]** execução do front-end (via docker) com você interagindo com a aplicação mostrando em quais momentos são feitas as chamadas aos outros componentes.*	De 60 a 90 segundos

Os cinco pontos apresentados **fazem parte do roteiro e são igualmente importantes** e diante da ausência ou incompletude de um deles haverá uma penalização de até 2,0 pts à nota final (até 0,4 pts por ponto). **A não entrega do vídeo implicará na penalização em 2,0 pts à nota final.**

Você deverá também disponibilizar seus componentes **em repositórios separados e públicos no GitHub**.

Sugestão: No momento da entrega publique o link completo (com https://...), evitando o uso de textos clicáveis (hyperlinks).

Sugestão de mensagem de entrega:
Olá, segue os dados referentes à entrega do meu MVP. *Link para o vídeo: https://www.youtube.com....* *Link para o repositório da componente principal: https://github.com/aluno123/minha_loja_front…* *Link para o repositório da segunda componente: https://github.com/aluno123/minha_loja_api…* *Foi utilizada a API 123, acessível pelo link: : https://api123.com/api*

Se tiver dúvidas sobre como criar um repositório público no GitHub, consulte:https://docs.github.com/pt/repositories/creating-and-managing-repositories/creating-a-new-repository