**Questão 1:**

**Front-end:**

O front-end refere-se à parte da aplicação que os usuários interagem diretamente. É a interface com a qual os usuários interagem visual e funcionalmente.

**HTML (Hypertext Markup Language):** HTML é a linguagem de marcação usada para estruturar o conteúdo da página, definindo elementos como títulos, parágrafos, imagens e links. Ele fornece a estrutura básica da página.

**CSS (Cascading Style Sheets):** CSS é usado para definir a apresentação e o estilo do conteúdo HTML. Ele permite que você defina cores, fontes, layout e outros aspectos visuais da página, tornando-a atraente e responsiva.

**Back-end**:

O back-end refere-se à parte da aplicação que lida com a lógica de negócios, processamento de dados, interações com o banco de dados e outras tarefas que acontecem nos bastidores.

**Node.js**: Node.js é uma plataforma que permite executar código JavaScript no servidor. Ele é amplamente utilizado para criar aplicativos web escaláveis e de alto desempenho. O Node.js permite que você construa a lógica do lado do servidor usando JavaScript.

**Python** (com framework Django): Python é uma linguagem de programação versátil e é frequentemente usada no desenvolvimento back-end. O framework Django é um conjunto de ferramentas poderosas para desenvolver aplicativos web em Python. Ele fornece muitas funcionalidades pré-construídas que aceleram o processo de desenvolvimento.

**Questão 2:**

**a) ControllerBase e Herança:**

No contexto do desenvolvimento web, um controlador é responsável por gerenciar as requisições vindas do cliente (navegador, aplicativo móvel, etc.) e coordenar a interação entre o usuário e o sistema. Ao criar um controlador, você está definindo a lógica para processar as requisições e retornar as respostas apropriadas.

Ao fazer uma herança de ControllerBase (ou de um equivalente, dependendo da estrutura do seu projeto), você está aproveitando as funcionalidades e convenções fornecidas pelo framework para facilitar o desenvolvimento. ControllerBase é uma classe base que normalmente inclui métodos e propriedades comuns usados nos controladores, como métodos para tratar requisições HTTP (GET, POST, etc.), manipulação de parâmetros e redirecionamento de ações.

**b) Chamada de Rotas GET e POST**:

A diferença fundamental entre as chamadas de rotas GET e POST reside na forma como os dados são transmitidos entre o cliente (navegador) e o servidor.

GET: A solicitação GET é usada para recuperar informações do servidor. Os parâmetros são anexados à URL como parte da query string, tornando-os visíveis na barra de endereços do navegador. Como a query string tem um limite de tamanho, as solicitações GET não são adequadas para enviar grandes quantidades de dados. Por serem visíveis e terem limitações, os navegadores permitem que você chame rotas GET diretamente na barra de endereços.

POST: A solicitação POST é usada para enviar dados para o servidor, que não são visíveis na URL. Os dados são incluídos no corpo da solicitação HTTP. Como os dados não são visíveis e o corpo da solicitação não tem um limite de tamanho como a query string, as solicitações POST são mais apropriadas para enviar dados confidenciais ou grandes quantidades de informações. Não é possível chamar rotas POST diretamente na barra de endereços do navegador, porque isso exigiria enviar dados no corpo da solicitação, o que não é possível somente com o navegador.

**c) Papel do Swagger:**

O Swagger é uma ferramenta de código aberto para projetar, criar, documentar e testar APIs (Interfaces de Programação de Aplicativos). Ele permite que os desenvolvedores descrevam suas APIs de maneira padronizada e gerem automaticamente documentação interativa a partir dessas descrições. A documentação gerada pelo Swagger é chamada de "Swagger UI".

O papel do Swagger é proporcionar uma maneira fácil e organizada de documentar APIs, permitindo que desenvolvedores e usuários entendam como usar as APIs, quais endpoints estão disponíveis, quais parâmetros são necessários, exemplos de solicitações e respostas, etc. Além disso, o Swagger UI oferece uma interface interativa que permite testar as APIs diretamente na documentação.

Usamos o Swagger porque ele facilita a comunicação entre desenvolvedores, torna mais fácil para terceiros entenderem e utilizarem nossas APIs, e também ajuda na validação e teste das APIs durante o desenvolvimento. Isso resulta em um processo de desenvolvimento mais eficiente e uma melhor experiência para os usuários que consomem as APIs.

**Questão 3:**

**a) Criar aula**

Verbo HTTP: POST

**b) Remover um material**

Verbo HTTP: DELETE

**c) Listar as disciplinas disponíveis para o professor no semestre**

Verbo HTTP: GET

**d) Listar os alunos da disciplina**

Verbo HTTP: GET

**e) Atualizar um material**

Verbo HTTP: PUT (ou PATCH, dependendo da necessidade)

**f) Lançar uma nota**

Verbo HTTP: POST (se for criar um recurso de nota)

**Questão 4:**

**Ok():**

Código de status HTTP: 200 OK

Significado: Indica que a requisição foi bem-sucedida e que o servidor retornou os dados solicitados.

Uso: Geralmente usado quando uma operação GET ou POST é bem-sucedida e os dados estão sendo retornados.

**BadRequest():**

Código de status HTTP: 400 Bad Request

Significado: Indica que a requisição do cliente é inválida, malformada ou não pode ser processada pelo servidor devido a erros nos parâmetros ou formato da requisição.

Uso: Usado quando os dados fornecidos pelo cliente não são válidos ou quando a requisição não segue as regras esperadas.

**NotFound():**

Código de status HTTP: 404 Not Found

Significado: Indica que o servidor não encontrou o recurso solicitado na URL especificada.

Uso: Usado quando o recurso solicitado não existe no servidor ou a URL está incorreta.

**Questão 5:**

**https://:** Indica que o protocolo de comunicação é seguro e criptografado.

**minhaapi.com.br:** É o domínio onde a API está hospedada, identificando o servidor.

**/api/vendas:** Representa o caminho da API, indicando que estamos acessando o recurso de vendas dentro da API.