Fundamentos de APIs

Protocolo HTTP

- HTTP é o protocolo base de comunicação na Web, incluindo a comunicação a recursos de APIs
- Cabeçalhos de HTTP permitem a atribuição de dados extras à requisição e correspondente resposta HTTP
- Entre os cabeçalhos mais comuns estão:
    - Content-Length
    - Content-Type
    - Authorization
    - Cache-Control
    - User-Agent

Protocolo HTTP

- Status de resposta de uma requisição HTTP é basicamente um código numérico que representa o resultado dela
- Os status HTTP mais comuns são:
  - **200 (OK):** sucesso, geralmente contém dados de retorno
  - **201 (Created):** sucesso, geralmente retorna uma representação do objeto criado, e a URL de detalhes no cabeçalho location
  - **202 (Accepted):** sucesso, indica que um processamento posterior vai ser executado no fluxo
  - **204 (No Content):** sucesso, sem dados de retorno
  - **400 (Bad Request):** erro, indica algum erro nos dados da requisição, como valores inválidos

Protocolo HTTP

- Os status HTTP mais comuns são:
  - **401 (Unauthorized):** erro, indica uma requisição que precisa ser autenticada
  - **403 (Forbidden):** erro, indica uma falta de permissão de acesso ao recurso solicitado
  - **404 (Not Found):** erro, indica que o recurso não foi encontrado
  - **500 (Internal Server Error):** erro não esperado

Protocolo HTTP

- Além do código de status, uma requisição HTTP é representada por uma ação realizada, chamada de método
- Os métodos mais comuns são:
  - **GET:** utilizado geralmente para consultas
  - **POST:** utilizado geralmente para cadastros
  - **PUT:** utilizado geralmente para atualizações
  - **DELETE:** utilizado geralmente para remoções / desativações
- O PATCH também é utilizado, mas não é tão comum
- Junto com o código de status, é uma das fontes de mais confusão e falta de alinhamento em integrações de sistemas
- É necessário seguir boas práticas para simplificar esse processo

Boas Práticas quanto a códigos de resposta e métodos HTTP

Códigos de Resposta e Métodos HTTP

- Respostas comuns por cada método HTTP
  - **GET:**
    - **Sucesso:** Ok
    - **Erro:** Not Found (recurso com identificador não encontrado, em caso de listas se retorna Ok mesmo se não tiver elementos)
  - **POST:**
    - **Sucesso:** Created
    - **Erro:** Bad Request (quando os dados são inválidos)
  - **PUT:**
    - **Sucesso:** No Content
    - **Erro:** Not Found (recurso com identificador não encontrado), Bad Request (quando os dados são inválidos)
  - **DELETE:**
    - **Sucesso:** No Content
    - **Erro:** Not Found (recurso com identificador não encontrado)

Padrão REST

Padrão REST

- REST, ou Representational State Transfer, é um padrão utilizado para definição de interfaces de comunicação entre sistemas, comumente utilizado com o protocolo HTTP
- Em suas definições o principal conceito é o de recurso, que representa um objeto que está sendo “explorado” em operações da interface
- Por exemplo, no DevFreela, alguns recursos a serem gerenciados são:
  - Projetos
  - Usuários
  - Habilidades
  - Comentários

Padrão REST

- Alguns exemplos de pontos de acesso (endpoints) de uma API REST
  - api/users (GET, POST)
  - api/projects (GET, POST)
  - api/projects/1 (GET, PUT, DELETE)
  - api/projects/1/comments (POST)

Até a próxima aula!

Introdução a ASP.NET Core

Introdução a ASP.NET Core

- Framework de código-aberto, multiplataforma, leve, e de alto desempenho para a construção de aplicações web modernas
- Lançado em junho de 2016, está na versão .NET 8 (LTS), e é a escolha padrão para novos projetos na plataforma .NET
- Tem recursos nativos como
  - Injeção de dependência
  - Middlewares
  - Configuração por ambiente
  - Serviços em segundo plano
- Excelente curva de aprendizagem

Introdução a ASP.NET Core

- Instalado junto ao Visual Studio 2022 e .NET SDK
  - Caso utilize o Visual Studio 2022, lembrar de selecionar a opção de desenvolvimento Web com ASP.NET, que irá instalar os requisitos necessários para o desenvolvimento
  - O Jetbrains Rider oferece uma opção para instalação do .NET SDK já em sua configuração
  - Já o Visual Studio Code necessita da instalação da .NET SDK de forma separada

Até a próxima aula!

Controllers e Actions

Controllers e Actions

- Controllers são classes que agrupam um conjunto de Actions, e que herdam de Controller ou ControllerBase
- Agrupam de maneira lógica baseado no recurso a ser acessado
- **Exemplo:** ProjectsController

- Actions são métodos que estão contidos nos Controller, e representam endpoints. Através delas são definidas as rotas e métodos HTTP utilizados, e seu tipo retorno geralmente é *IActionResult*, implementado por respostas como Ok, NotFound, entre outras

Controllers e Actions

- Importante separar modelos de dados de entrada e saída, do modelo de domínio
- Com isso, evitamos exposição de dados confidenciais que devem ficar apenas na entidade e não no modelo de saída
- Modelos de entrada e saída são basicamente DTOs, Data Transfer Objects, e são chamados respectivamente de Input Models e View Models
    - Reforçando: DTO é um termo genérico

Até a próxima aula!

Aprofundando em Rotas

Aprofundando em Rotas

- O ASP.NET Core é bem flexível a respeito de definição de Actions e suas rotas, além da recepção de parâmetros
- Para uma requisição GET, por exemplo, geralmente você vai receber
  - **Parâmetro de URL**
    - meusite.com.br/api/projects/1234
  - **Query String**
    - meusite.com.br/api/projects?search=abc

Aprofundando em Rotas

- Já para uma requisição POST e PUT, por exemplo, geralmente você vai receber
  - **Parâmetro de URL**
    - meusite.com.br/api/projects/ POST
    - meusite.com.br/api/projects/1234 PUT
  - **Corpo da Requisição**
    - Dados em Formato JSON
    - { "title": “Projeto ABC”, "description": “Descrição do Projeto”, “totalCost”: 12000, “idClient”: 1, “idFreelancer”: 2}
  - Também é possível receber arquivos, com **form data**

Até a próxima aula!

Injeção de Dependência

Injeção de Dependência

- Técnica de extrair a responsabilidade de instanciar uma dependência para fora de uma classe, passando geralmente uma instância do objeto através de construtor
- Entre os benefícios está a melhoria de testabilidade, além de diminuição do acoplamento (quando utilizada com interfaces)

Injeção de Dependência

- No ASP.NET Core existem três tempos de ciclo de vida de um objeto na injeção de dependência:
  - **Singleton:** a mesma instância do objeto é utilizada em todo o escopo da aplicação, mesmo entre requisições diferentes.
  - **Scoped:** é utilizada uma instância para a requisição inteira.
  - **Transient:** é utilizada uma instância por cada uso;

Até a próxima aula!