Desenvolvimento Web com .NET C#

Construindo Web APIs Modernas

Curso Prático de ASP.NET Core Web API

Outubro 2025

Web APIs

Introdução ao Desenvolvimento Web

HTTP em 3 minutos

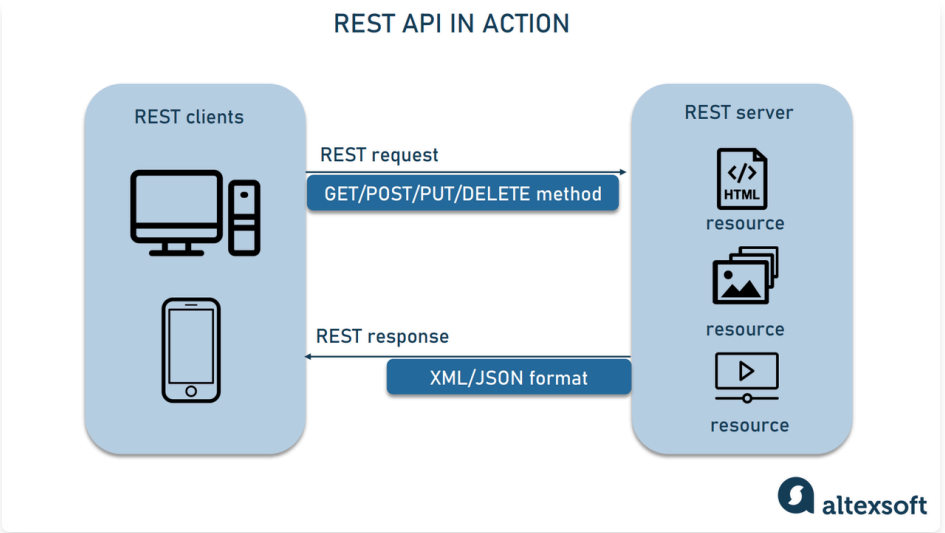
- Request/Response, Headers, Body
- Verbos: GET, POST, PUT, DELETE
- Status Codes: 2xx (sucesso), 4xx (erro cliente), 5xx (erro servidor)

REST

- Recursos identificados por URIs
- Interface uniforme, stateless, cacheable

APIs e Setup

- API: contrato HTTP para integração entre sistemas
- Setup: `dotnet new webapi`, Swagger
- Boas práticas: versionar API, HTTPS, ProblemDetails



Fluxo de requisição e resposta em uma API REST

Exercício

1) Crie a solução e o projeto Web API chamado "BankSystem.Api"

2) Habilite Swagger e execute a API localmente

3) Garanta rota base /api e teste um endpoint padrão no Swagger

Controllers e Rotas (MVC — foco em Controller)

MVC (visão geral)

- **Model**: representa o domínio e regras de negócio
- **View**: em APIs, são os dados formatados (JSON/XML)
- **Controller**: orquestra requisições, interage com modelos

Controllers em Web API

- Classe com atributos `[ApiController]`
- Rota base: `[Route("api/[controller]")]`
- Nome do arquivo: `NomeRecursoController.cs`

Rotas e convenções

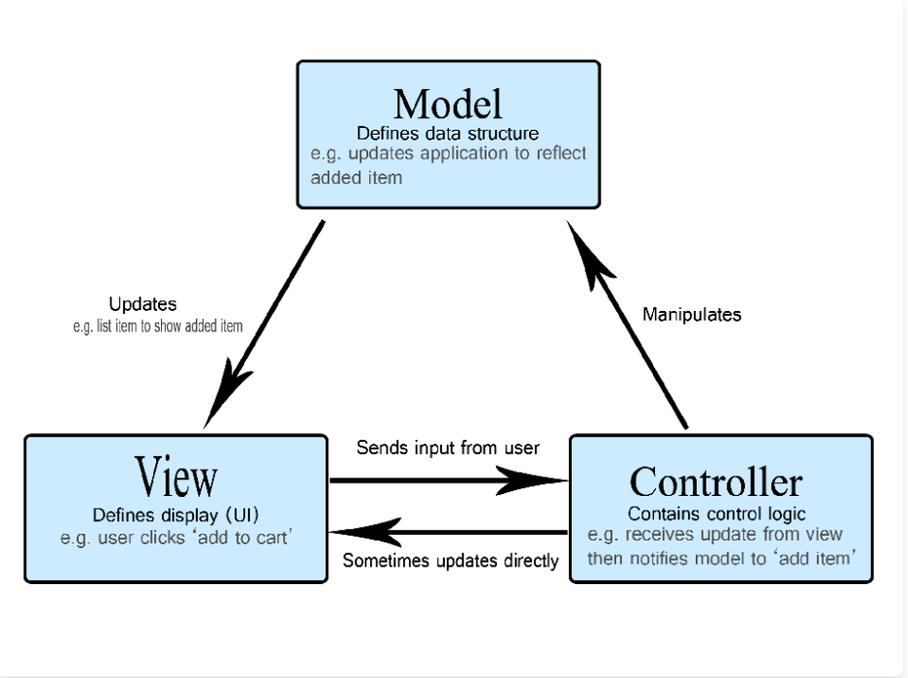
- Verbos por atributos: `[HttpGet]`, `[HttpPost]`, etc.
- Convenção plural para recursos (ex.: `/api/contas`)
- Testáveis via Swagger e Postman

Exercício

1) Crie o `ContasController` com rota base `/api/contas`

2) Adicione uma ação GET para retornar a lista de contas (use lista em memória)

3) Teste o endpoint no Swagger



Padrão MVC: separação de responsabilidades entre Model, View e Controller

DTOs (Data Transfer Objects) e Validação

Por que usar DTOs?

- Encapsula dados de entrada/saída da API
- Evita expor entidades de domínio diretamente
- Facilita versionamento, segurança e documentação

Tipos de DTOs

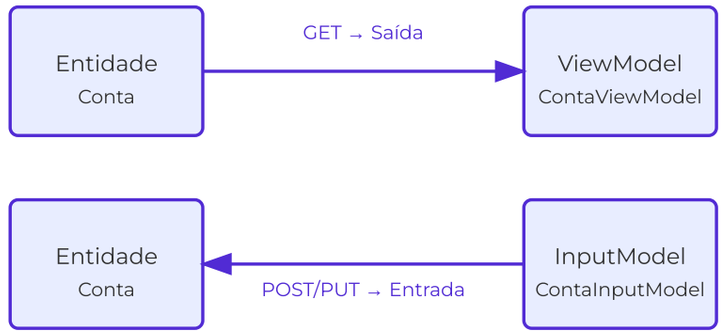
- **InputModel:** dados de entrada (POST/PUT)
- **ViewModel:** dados de saída (GET)

Validação com DataAnnotations

- `[Required], [StringLength], [Range]`
- `[ApiController]` ativa validação automática (400)

Mapeamento

- Manual: conversão explícita entre classes
- Automático: biblioteca AutoMapper



Fluxo de DTOs em uma API REST

Exercício

1) Crie ContaInputModel com validações básicas para número, titular e saldo

2) Crie ContaViewModel para retornar dados ao cliente

3) Prepare o endpoint POST /api/contas para receber InputModel e retornar ViewModel

Verbos HTTP - GET e POST

GET - Leitura de Recursos

- Apenas para consulta e leitura de recursos
- Parâmetros: Rota `[FromRoute]` e Query String `[FromQuery]`
- Status codes comuns: 200 (OK), 404 (Not Found)

POST - Criação de Recursos

- Usado para criar novos recursos no servidor
- Dados enviados no corpo da requisição `[FromBody]`
- Status code: 201 (Created) + cabeçalho Location

Mapeamento de Rotas

- `[HttpGet("{numero}")]` para recurso específico
- `[HttpPost]` na coleção para criar novo
- Idempotência: GET é idempotente, POST não

Exercício

1) Implemente GET /api/contas/{numero} para buscar conta por número

2) Implemente POST /api/contas para criar nova conta em memória

3) Retorne 201 com Location no POST e 404 quando não encontrar

HTTP Verbo	Operação CRUD	Status Code
GET	Read (Leitura)	200, 404
POST	Create (Criação)	201, 400
PUT	Update (Atualização)	200, 204, 404
DELETE	Delete (Remoção)	204, 404

```
// Exemplo de controller com GET e POST
[Route("api/[controller]")]
[ApiController]
public class ContasController : ControllerBase {
  [HttpGet("{numero}")]
  public ActionResult<ContaViewModel> Get(string
numero) {...}

  [HttpPost]
  public ActionResult Post(ContaInputModel model)
{...}
}
```

Verbos HTTP - PUT e DELETE

PUT

- Atualização completa de recurso ou operação idempotente
- Mesmo input = mesmo resultado (idempotência)
- Status Codes: 200 OK (com resposta), 204 No Content (sem corpo)

DELETE

- Remoção de recurso do servidor
- Normalmente retorna 204 No Content
- Pode retornar 200 OK se incluir informação de confirmação

Status Codes e Boas Práticas

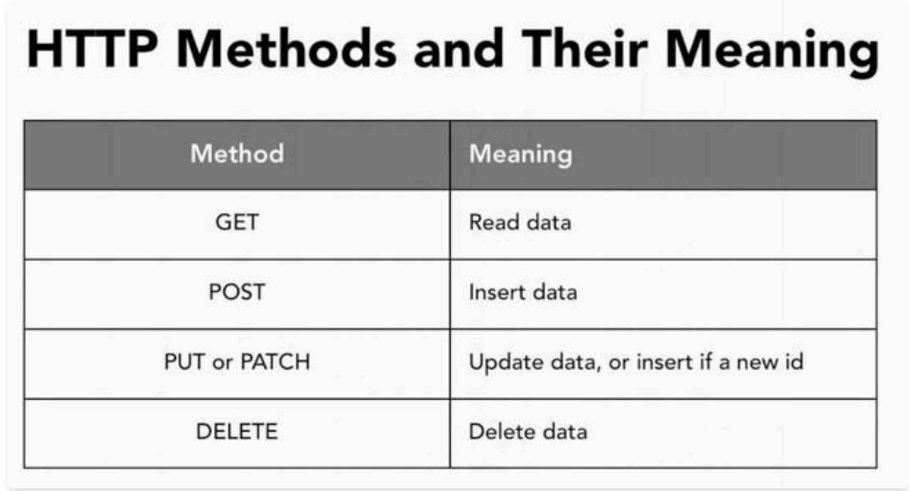
- 400 Bad Request: dados inválidos ou validação falhou
- 404 Not Found: recurso não encontrado
- 409 Conflict: violação de regra de negócio
- Validar regras de negócio antes da operação
- Retornar mensagens claras e contratos estáveis

Exercício

1) Crie PUT /api/contas/{numero}/deposito para acrescentar saldo

2) Implemente validações (saldo positivo, conta existente)

3) Crie DELETE /api/contas/{numero} e retorne status codes adequados



Verbos HTTP: PUT vs POST, DELETE e suas principais características

Uso adequado dos Status Codes

2xx	4xx	5xx
Sucesso	Erro do cliente	Erro do servidor