semana 9- aula 01
Linguagens de programação back-end
RESTful APIs
Código da aula: [SIS]ANO2C2B2S9A1

Objetivos da Aula:

- ❖ Compreender e implementar APIs RESTful.
- ❖ Exercitar a curiosidade ao explorar o funcionamento de APIs RESTful.

Exposição

Definição:

APIs RESTful são interfaces de comunicação que seguem os princípios arquiteturais do REST (Representational State Transfer). Elas utilizam métodos HTTP como GET, POST, PUT e DELETE para manipular recursos em um servidor. Cada recurso é identificado por uma URI (Uniform Resource Identifier) e é representado em formatos como JSON ou XML.

Os princípios fundamentais de uma API RESTful incluem:

- Statelessness – Cada requisição de cliente para o servidor deve conter todas as informações necessárias para ser processada;
- Representações de recursos – Os recursos são representados em formatos simples e legíveis, como JSON;
- Uniform interface – A comunicação entre cliente e servidor segue padrões bem definidos de uso de métodos HTTP.
- Temas Principais:
    1. Introdução a APIs RESTful: Conceitos, princípios e arquitetura REST.
    2. Métodos HTTP em APIs RESTful:
        - `GET`: Utilização para recuperar recursos (listar produtos, obter detalhes de um produto). Benefícios: idempotência, cacheamento.
        - `POST`: Utilização para criar novos recursos (adicionar um novo produto). Funcionamento no servidor e código de status (ex: 201 Created).
        - `PUT`: Utilização para atualizar um recurso existente de forma completa.
        - `DELETE`: Utilização para remover um recurso. Comportamento ao tentar remover recurso inexistente (ex: status 404 Not Found).
    3. Recursos e URIs: Identificação de recursos através de URIs (Uniform Resource Identifiers) e manipulação por meio de rotas específicas.
    4. Formato de Dados: Utilização de JSON para enviar e receber informações em requisições e respostas.
- Resumo e Aprendizados: Funcionamento dos métodos HTTP (GET, POST, PUT, DELETE) para manipulação de recursos, identificação de recursos por URIs e o uso de JSON como formato de dados padrão.

semana 9- aula 02
Linguagens de programação back-end
GraphQL
Código da aula: [SIS]ANO2C2B2S9A2

Objetivos da Aula:

- ❖ Compreender e implementar APIs RESTful.
- ❖ Desenvolver APIs utilizando GraphQL.

Exposição:

GraphQL é uma linguagem de consulta para APIs que viabiliza ao cliente especificar exatamente quais dados deseja recuperar do servidor, ao contrário das APIs RESTful, em que o servidor define os dados retornados em cada endpoint. Em GraphQL, existem três componentes principais:

- Queries: usadas para solicitar dados;
- Mutations: usadas para modificar dados;
- Schemas: definem a estrutura dos dados disponíveis na API, estabelecendo os tipos e as relações entre os dados.

Uma das principais vantagens de GraphQL é a capacidade de evitar over-fetching (quando o cliente recebe mais dados do que o necessário) e under-fetching (quando o cliente precisa fazer múltiplas requisições para obter todos os dados necessários).

- Temas Principais:
    1. Introdução ao GraphQL: Conceito e principais diferenças em relação a APIs RESTful.
    2. Queries em GraphQL:
        - Recuperação de dados.
        - Capacidade do cliente de especificar exatamente quais campos deseja (evitando *over-fetching* e *under-fetching*).
        - Comparação com requisições `GET` em REST.
    3. Mutations em GraphQL:
        - Modificação de dados no servidor (criar, atualizar, deletar).
        - Diferenciação em relação às *queries*.
        - Exemplos práticos de implementação de uma *mutation*.
    4. Schema em GraphQL:
        - Definição da estrutura da API (tipos de dados, *queries* e *mutations* disponíveis).
        - O schema como um contrato entre cliente e servidor.
- Resumo e Aprendizados: Vantagens do GraphQL em especificar dados (evitando *over/under-fetching*), distinção e uso de *queries* e *mutations*, e o papel central do *schema*.

semana 9- aula 03
Linguagens de programação back-end
Segurança em APIs
Código da aula: [SIS]ANO2C2B2S9A3

Objetivos da Aula:

- ❖ Implementar práticas de segurança em APIs.

Exposição:

As APIs são pontos de entrada para sistemas e, portanto, alvos frequentes de ataques. Implementar práticas de segurança robustas é crucial para proteger os dados e os recursos do servidor. Alguns dos principais conceitos de segurança em APIs incluem:

- Autenticação – Verificar a identidade de um usuário ou sistema que está tentando acessar a API. Os métodos mais comuns incluem JWT (JSON Web Tokens) e OAuth;
- Autorização – Controlar o que um usuário autenticado pode fazer, garantindo que eles possam acessar ou modificar apenas os recursos aos quais têm permissão;
- Rate limiting – Limitar o número de requisições que um cliente pode fazer dentro de um determinado período, protegendo a API contra-ataques de negação de serviço (DoS);
- Criptografia – Garantir que os dados enviados e recebidos pela API estejam criptografados, geralmente utilizando HTTPS;
- Validação de dados – Conferir se os dados enviados pelo cliente são válidos e dentro das expectativas, prevenindo injeções de código ou outros ataques maliciosos.
- Autenticação com JWT – A rota/o login garante que o usuário se autentique enviando um nome de usuário e uma senha. Se as credenciais forem válidas, um JWT (token) é gerado e enviado para o cliente, que deve incluir esse token em todas as requisições subsequentes;
- Autorização com JWT – As rotas protegidas (/produtos, /produtos/<id>) utilizam o decorador @token_requerido para garantir que apenas usuários autenticados possam acessar os dados. Apenas o usuário "admin" pode adicionar ou remover produtos;
- Validação de tokens – O token é validado em cada requisição. Se for inválido ou expirado, o acesso é negado.

- Temas Principais:
    1. Autenticação e Autorização em APIs: Conceitos fundamentais.

2. JWT (JSON Web Tokens):
    - Utilização para autenticar usuários e autorizar o acesso a recursos protegidos.
    - Estrutura de um JWT e como é assinado digitalmente.
    - Envio do JWT no cabeçalho das requisições.
3. Validação de Tokens:
    - Importância de validar o token em cada requisição.
    - Procedimentos em caso de token expirado ou inválido (negação de acesso).
4. Implementação Prática de Segurança:
    - Criptografia de tokens (conceito).
    - Validação de dados de entrada.
    - Uso de decoradores (ex: `@token_requerido` em Python/Flask) para proteger rotas.
    - Função `@wraps` para preservar metadados da função original ao usar decoradores.

- Resumo e Aprendizados: Uso de JWT para autenticação e autorização, importância da validação contínua de tokens, e implementação de práticas como criptografia e validação de dados para proteger APIs.

As diferenças entre APIs RESTful e GraphQL

Aqui está um quadro comparativo para destacar as principais diferenças:

Quadro Comparativo: RESTful APIs vs. GraphQL

Característica	APIs RESTful	GraphQL
Estrutura de Dados	Retorna uma estrutura de dados fixa por endpoint.	O cliente especifica exatamente os dados que precisa.
Endpoints	Múltiplos endpoints (URIs) para diferentes recursos.	Tipicamente um único endpoint para todas as interações.
Busca de Dados	Pode levar a:	Evita:

	- Over-fetching (receber mais dados que o necessário).	- Over-fetching
	- Under-fetching (precisar de múltiplas chamadas para obter todos os dados).	- Under-fetching (pode buscar múltiplos recursos em uma única chamada).
Métodos HTTP	Utiliza diversos métodos HTTP (GET, POST, PUT, DELETE, etc.) para operações distintas em recursos.	Geralmente usa POST para todas as operações (queries e mutations), com a operação especificada no corpo da requisição.
Tipagem	Não possui um sistema de tipos fortemente integrado no protocolo em si.	Fortemente tipado através de um *Schema Definition Language* (SDL). O schema define os tipos de dados e operações possíveis.
Versionamento	Frequentemente versionado via URI (ex: `/v1/users`) ou cabeçalhos.	Menos necessidade de versionamento explícito, pois os clientes solicitam apenas os campos que conhecem. Campos podem ser marcados como obsoletos (*deprecated*).
Estado	Geralmente *stateless* (sem estado).	Geralmente *stateless* (sem estado).
Descoberta	Pode usar padrões como HATEOAS, mas não é mandatório.	O schema serve como uma documentação viva e permite introspecção para descobrir os tipos e campos disponíveis.

Complexidade no Cliente	Pode precisar de lógica para agregar dados de múltiplas chamadas.	Simplifica a busca de dados complexos e relacionados.
Complexidade no Servidor	Mais simples para endpoints que mapeiam diretamente para modelos de dados.	Pode exigir mais lógica no servidor para resolver queries complexas e otimizar o acesso aos dados.
Formato de Resposta	Comumente JSON, mas pode ser XML, texto, etc.	Comumente JSON. A estrutura da resposta espelha a estrutura da query.

Principais Diferenças Explicadas:

1. Como os dados são solicitados e o que é retornado:
    - RESTful: Você solicita um recurso específico (ex: `/users/123`), e o servidor decide quais informações sobre esse usuário retornar. Se você quiser apenas o nome e o email, mas o endpoint retorna o nome, email, endereço e histórico de compras, você recebe tudo (over-fetching). Se você precisa de informações do usuário e seus posts recentes (que estão em outro endpoint, ex: `/users/123/posts`), você precisa fazer duas requisições (under-fetching).
    - GraphQL: O cliente envia uma "query" que especifica exatamente quais campos de quais recursos ele quer. Por exemplo, você pode pedir o `nome` e `email` do usuário com ID 123, e também os `titulos` dos seus últimos 3 `posts`, tudo em uma única requisição para um único endpoint. O servidor retornará apenas esses dados.
2. **Número de Endpoints:**
    - RESTful: Cada recurso ou coleção de recursos geralmente tem seu próprio endpoint (URI). Ex: `/users`, `/posts`, `/products`.
    - GraphQL: Normalmente, expõe um único endpoint (ex: `/graphql`). Todas as queries, mutations (modificações de dados) e subscriptions (dados em tempo real) são enviadas para este endpoint.
3. **Sistema de Tipos (Schema):**
    - RESTful: Não há um sistema de tipos formalizado e obrigatório como parte do padrão REST em si. A estrutura dos dados é definida pela implementação do servidor.

    - GraphQL: É construído em torno de um schema fortemente tipado. O schema define todos os tipos de dados que a API pode retornar e todas as operações (queries, mutations) que podem ser executadas. Isso serve como um contrato entre o cliente e o servidor e permite ferramentas poderosas de desenvolvimento e validação.
4. **Métodos de Operação:**
    - RESTful: Utiliza os verbos HTTP (GET para buscar, POST para criar, PUT/PATCH para atualizar, DELETE para remover) para indicar a intenção da operação sobre um recurso específico.
    - GraphQL: As operações são definidas como `query` (para leitura), `mutation` (para escrita/modificação) ou `subscription` (para dados em tempo real). Essas operações são geralmente enviadas via HTTP POST no corpo da requisição.

Vídeo Explicativo:

https://www.youtube.com/watch?v=dsarexwqcjc