P.PORTO

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Introdução Web APIs

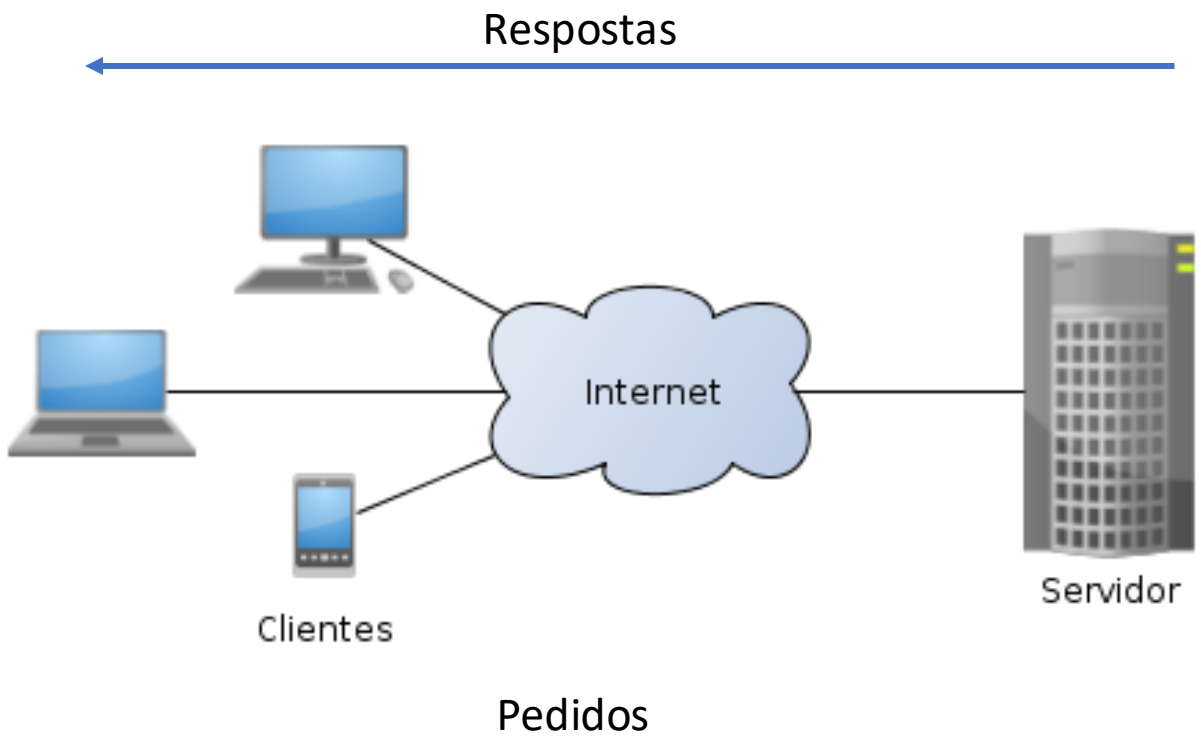
Bruno Oliveira (bmo@estg.ipp.pt)

PEI Processamento Estruturado de Informação

Arquitetura Cliente-Servidor

- A comunicação entre um browser e um site utiliza a arquitetura Cliente-Servidor.
- Os clientes (por exemplo web browser) realizam pedidos de forma a requisitar recursos disponibilizados por servidores web;
- Esta troca de mensagens é efetuada de acordo com o protocolo HTTP, que é a base de qualquer troca de dados na web.

Arquitetura Cliente-Servidor



Arquitetura Cliente-Servidor

- Os dados são trocados entre computadores através de “packets” que descrevem o formato com que os dados são enviados;
- O browser (por exemplo) é responsável por reunir todos os pacotes num website e apresentar o seu conteúdo.

O que é o pedido HTTP?

- O mensagem do pedido contém:
  - Request-line: identifica o Request method, o recurso a solicitar e a versão do HTTP;
  - Request method: GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS e TRACE, indicando o método de enviado da informação para o servidor;

O que é o pedido HTTP?

- O mensagem do pedido contém:
  - Request Header: contém informação adicional sobre o pedido, como por exemplo: preferências de idioma ou o tipo de cliente utilizado;
  - Request Body: contém conteúdo que será enviado para o servidor, utilizando por exemplo XML ou JSON.

O que é o pedido HTTP?

Linha de requisição

Linhas de cabeçalho

Corpo

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://clearforest.com/">string</string>
```

Exemplo modificado de: https://www.tutorialspoint.com/http/http_requests.htm

Tipos de resposta HTTP

- O mensagem da resposta contém:
  - Message Status-Line: consiste na identificação da versão do protocolo HTTP, um código de estado e uma identificação textual do estado;
  - Status Code: 3 dígitos to tipo inteiro em que o primeiro dígito identifica a categoria do estado associado à resposta: 1XX – informação, 2XX- Sucesso, 3XX- redireccionamento, 4XX – Erro por parte do cliente, 5XX – Erro por parte do servidor;
  - Response Header Fields: informação adicional sobre a resposta relacionada com o servidor ou com pedidos futuros;

O que é a resposta HTTP?

MIME types: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Basico_sobre_HTTP/MIME_types

Exemplo modificado de: https://www.tutorialspoint.com/http/http_requests.htm

Serviços Web

- Serviços são componentes Web que podem ser publicados, encontrados e utilizados através da internet;
- Representam a comunicação entre aplicações através da internet;
- Assumem um papel crucial na troca de dados entre sistemas, principalmente entre sistemas heterógenos onde a complexidade de comunicação é mais significativa;

Serviços SOAP

- SOAP (Simple Object Access Protocol) é um protocolo de comunicação que define o formato para a troca de mensagens com um serviço web;
- Uma mensagem SOAP é um documento XML com os seguintes elementos:
  - Um elemento: Envelope que identifica o documento XML como uma mensagem SOAP;
  - Um elemento: Header que contém informação do cabeçalho;
  - Um elemento: Body que contém informação sobre as mensagens;
  - Um elemento: Fault que contém informação de erro e estado;

Serviços SOAP

Serviços REST

- REST representa um estilo de arquitetura que tem por base o protocolo HTTP;
- Numa arquitetura REST são disponibilizados recursos que é acedido através de uma interface comum baseado em métodos HTTP standard;
- É utilizado um servidor que disponibiliza o acesso aos recursos, enquanto que o cliente REST pode aceder e modificar os recursos REST;

Serviços REST

- Um pedido HTTP é equivalente a uma invocação de um método (operação) de um recurso disponível no servidor;
- O método HTTP é utilizado para determinar a operação a realizar sobre um determinado recurso;
- Cada recurso é identificado por URIs (Uniform Resource Identifier) e deverá suportar as operações HTTP;

Serviços REST

- Os parâmetros podem ser enviados no URL e/ou no corpo do pedido;
- Os tipos de dados utilizados no pedido e na resposta devem ser acordados entre o servidor e o(s) cliente(s);
- JSON e XML estão entre os tipos mais utilizados.

SOAP ou REST?

- Vantagens SOAP:
  - SOAP é mais adequado para sistemas mais complexos e que requeiram uma segurança mais robusta.
  - Além disso, também fornece um sistema de mensagens de falha integrado.
- Vantagens REST:
  - REST é mais simples, flexível e rápido.
  - Além disso, permite uma maior diversidade de formatos de dados e pode recorrer a um maior número de métodos HTTP.

REST Web API

- Uma (web) API representa uma interface web para o software;
- Pode ser visto como um contrato entre um fornecedor de dados e um consumidor estabelecendo o conteúdo exigido pelo consumidor (o pedido) e o conteúdo exigido pelo fornecedor (a resposta).
- Por exemplo, o design da API para um serviço de meteorologia pode exigir que o utilizador forneça um código postal e que o fornecedor responda com as devidas temperaturas.

REST Web API

- Com REST, os recursos são solicitados como se tratassem de uma caixa negra em que os pormenores de implementação não são conhecidos;
- REST é stateless, i.e., nenhum estado é mantido entre execuções do serviço.

Exemplo de Web API

- Existem muitas APIs públicas que podem ser utilizadas e integradas em aplicações. Exemplos: https://github.com/public-apis/public-apis
- A AmiiboAPI é uma RESTful API que permite consultar informação acerca de amiibos:

https://www.amiiboapi.com/api/amiibo/?character=zelda

endereço *recurso* parâmetros

REST – API Call

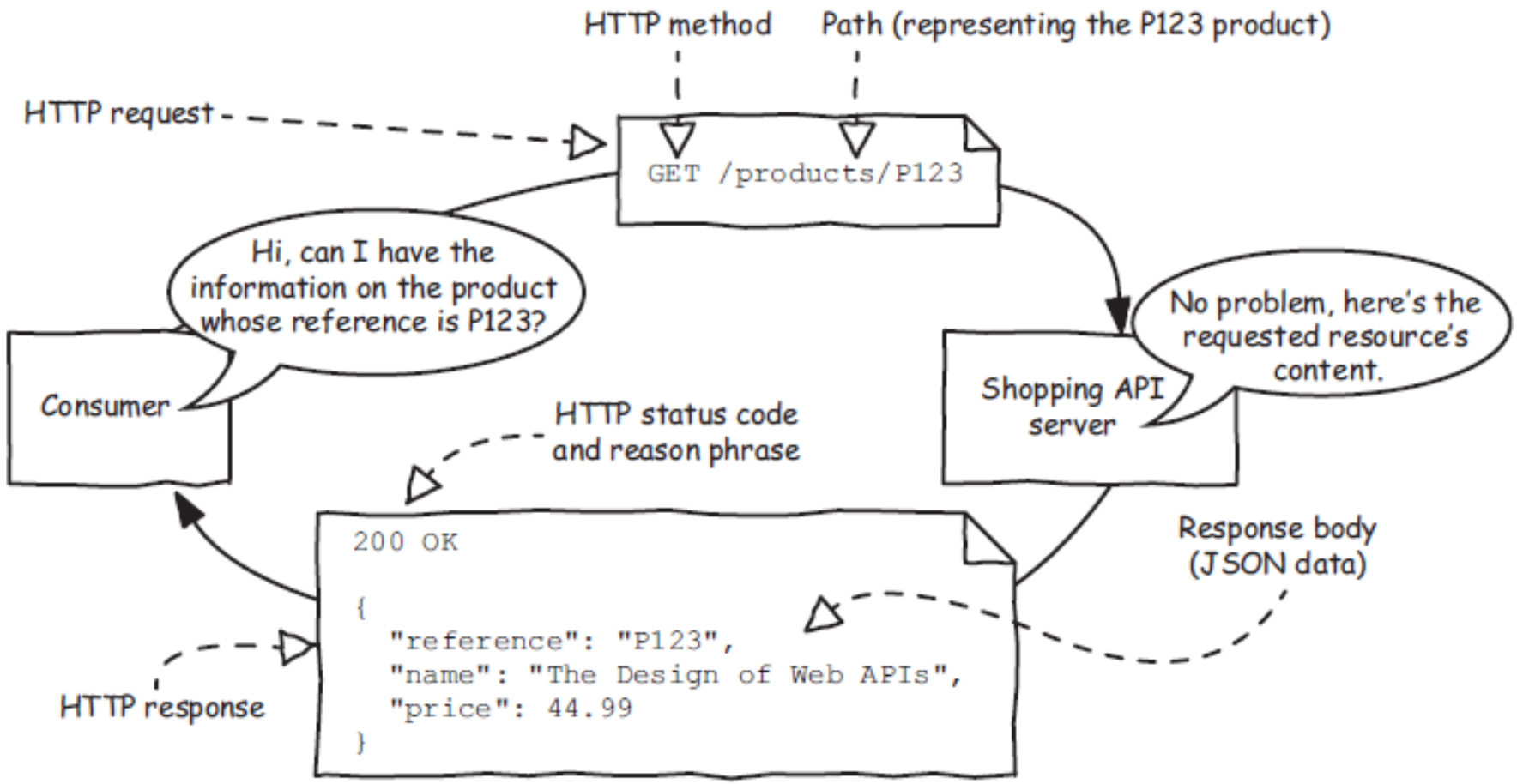


Imagem retirada do livro:Arnaud Lauret - The Design of Web APIs

Princípios básicos de uma REST API

- Uma API REST depende totalmente do protocolo HTTP;

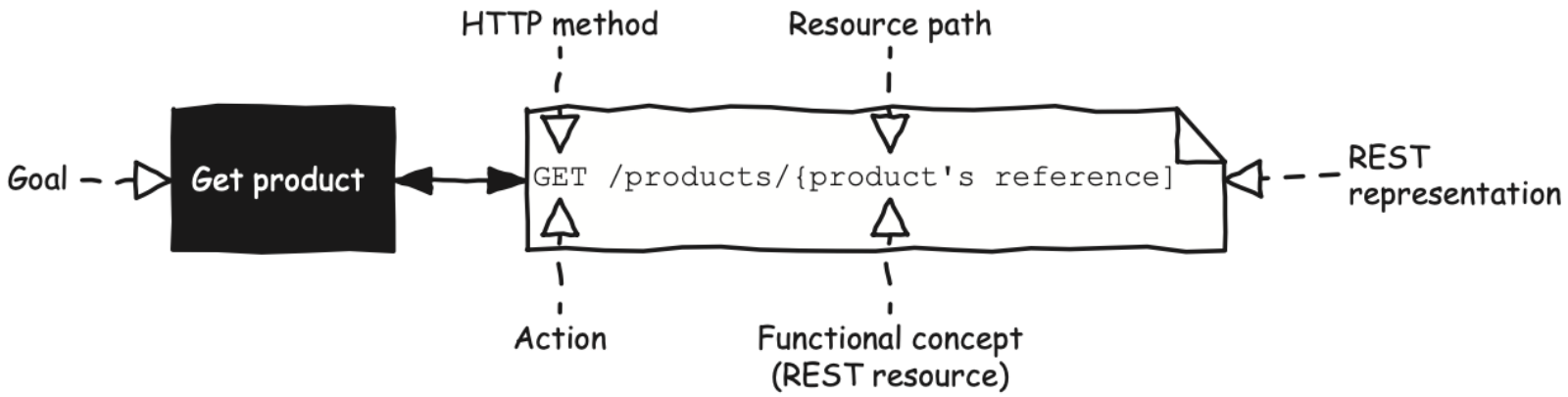


Imagem retirada do livro:Arnaud Lauret - The Design of Web APIs

Representar ações com HTTP

- Quando adicionamos um produto a um catálogo identificado por /products, criamos um resource de produto:

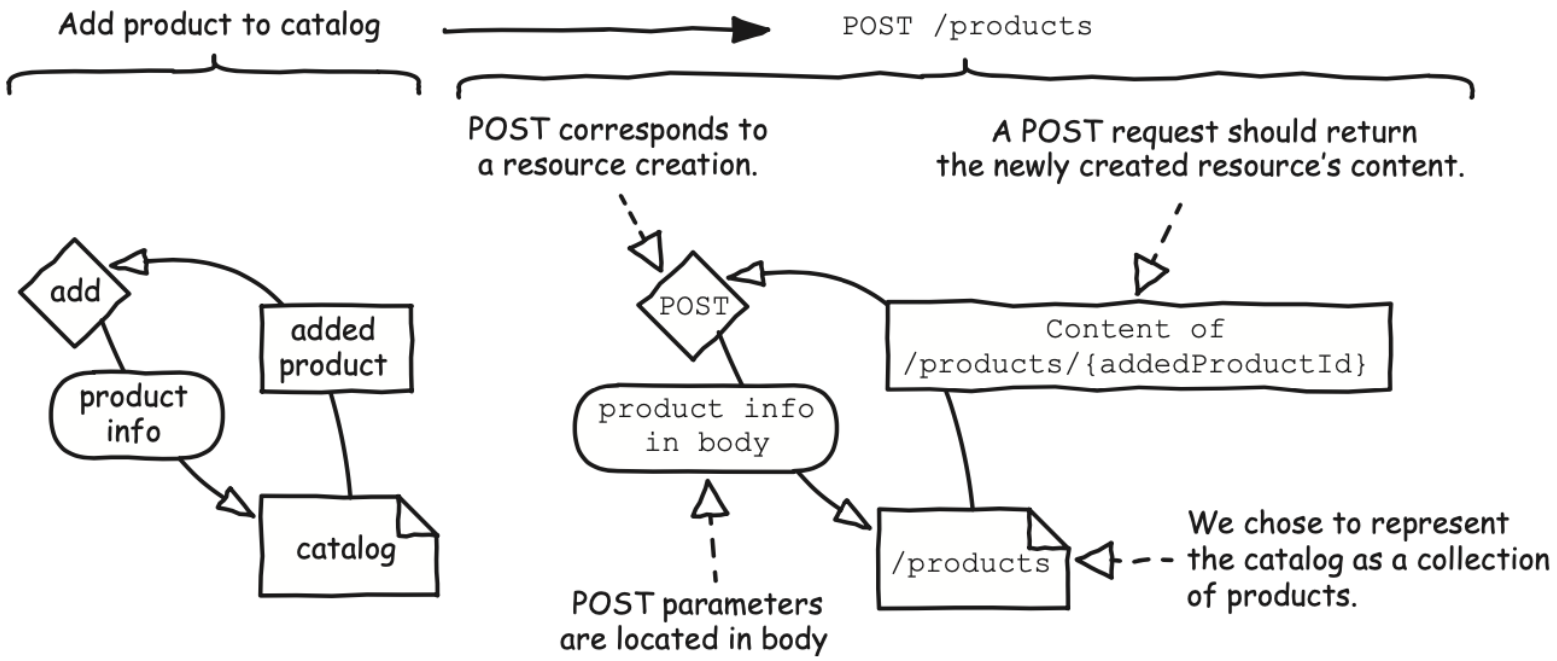


Imagem retirada do livro:Arnaud Lauret - The Design of Web APIs

Representar ações com HTTP

- A representação HTTP da pesquisa de produtos no catálogo usando uma consulta livre é GET.

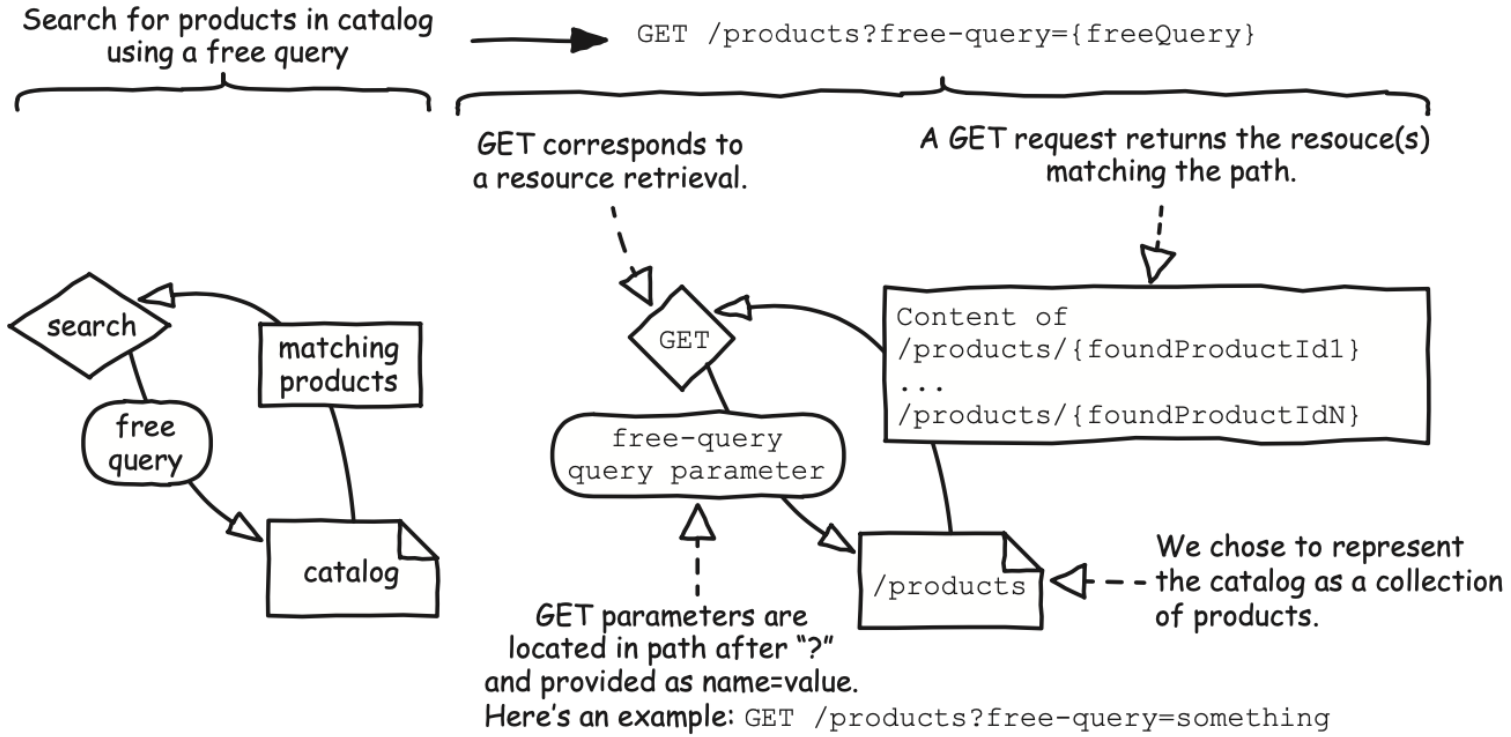


Imagem retirada do livro:Arnaud Lauret - The Design of Web APIs

Representar ações com HTTP

- A representação HTTP para a “eliminação” de um produto no catálogo é DELETE.

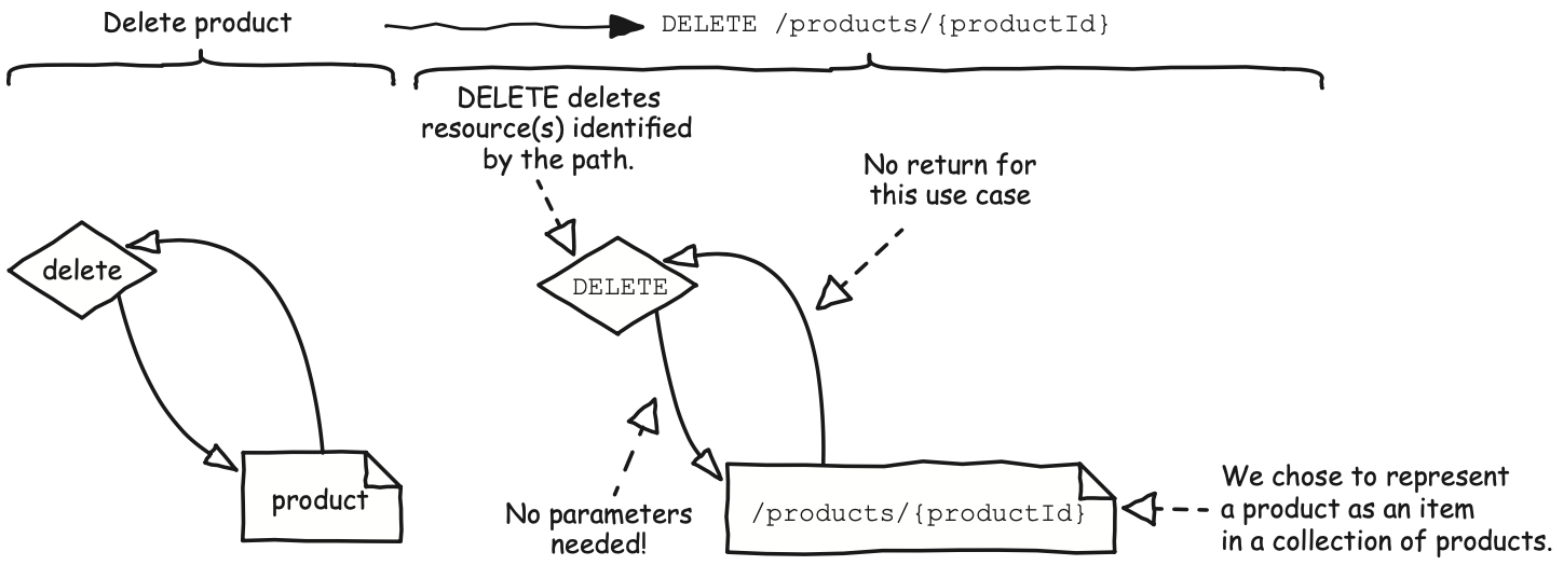


Imagem retirada do livro:Arnaud Lauret - The Design of Web APIs

Representar ações com HTTP

- A representação HTTP para a atualização de um produto no catálogo é PATCH.

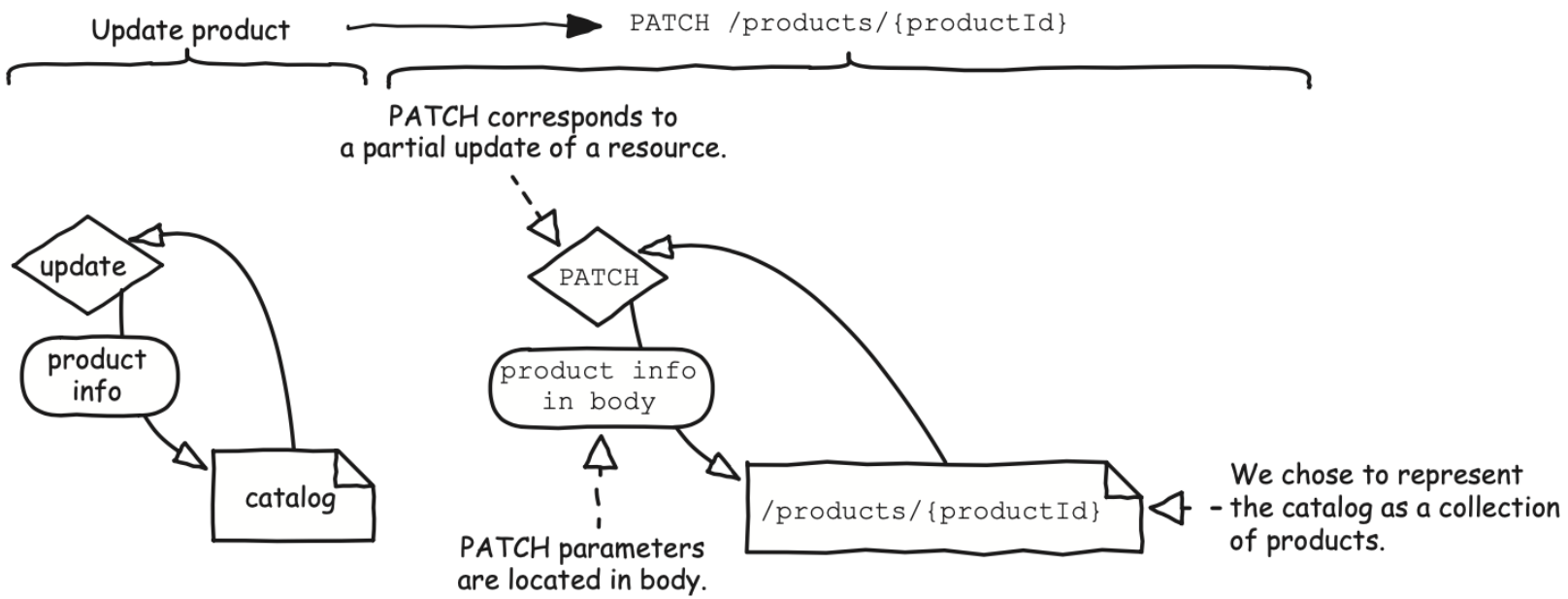


Imagem retirada do livro:Arnaud Lauret - The Design of Web APIs

Representar ações com HTTP

- O método PUT pode ser usado para substituir totalmente um recurso existente ou para criar um inexistente e fornecer seu identificador

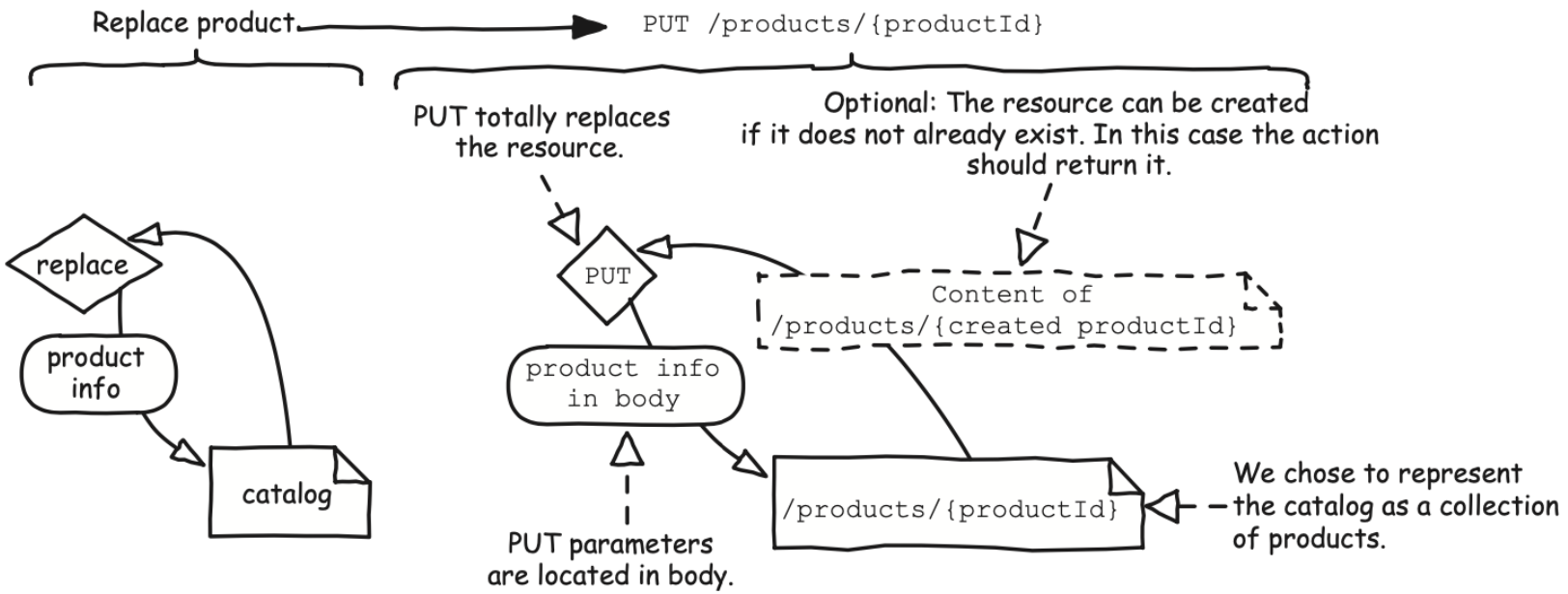


Imagem retirada do livro:Arnaud Lauret - The Design of Web APIs

Representar ações com HTTP

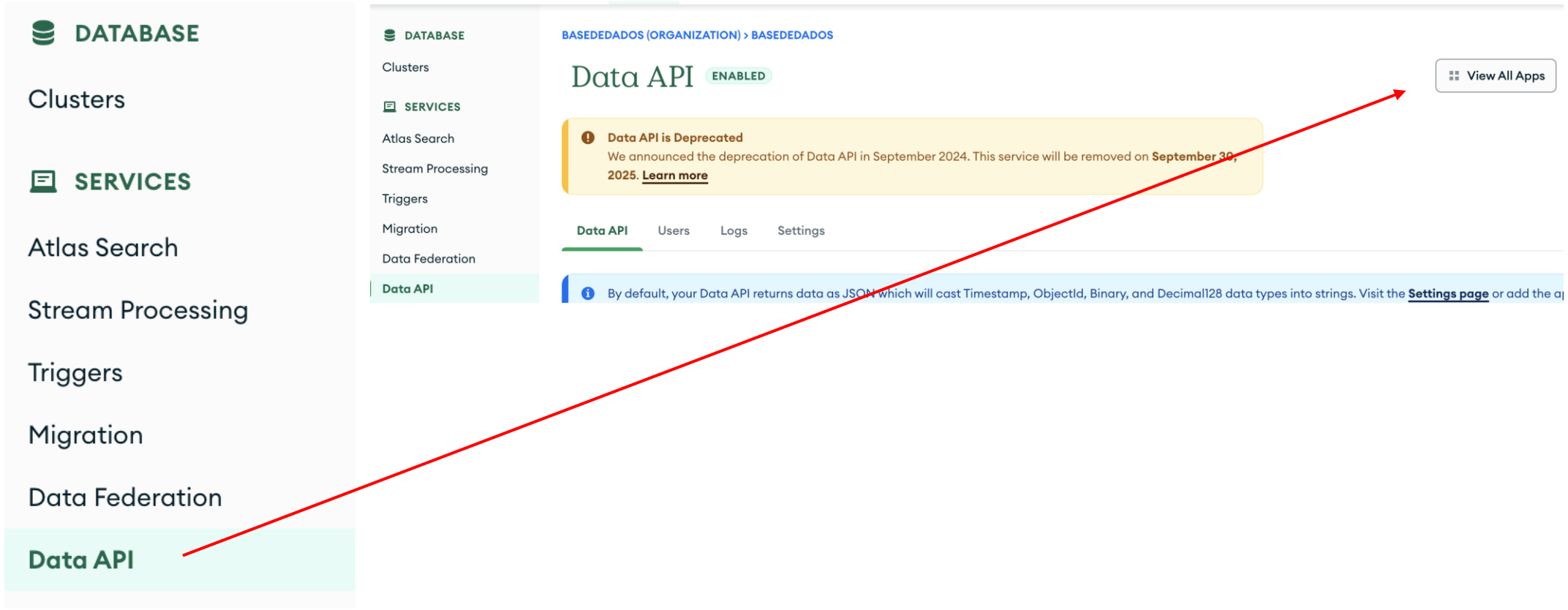
- Estas ações são realizadas do ponto de vista do consumidor!

Método HTTP	Ação
POST / PUT se for utilizado para criar recursos	Criar um cliente, adicionar uma refeição a um menu, enviar uma mensagem para o atendimento ao cliente, assinar um serviço, abrir conta bancária, upload de uma foto, (…)
GET	Ler um cliente, procurar um restaurante francês, encontrar novos amigos, recuperar contas abertas nos últimos 3 meses, download um contrato assinado, filtrar livros mais vendidos, selecionar fotos, listar amigos, (…)
PATCH/PUT	Atualizar um cliente, substituir mercadorias num pedido, trocar de assento de avião, editar o método de entrega de um pedido, alterar a moeda de um pedido, modificar um limite de cartão de débito, bloquear temporariamente um cartão de crédito, (…)
DELETE	Excluir um cliente, cancelar um pedido, fechar um caso, encerrar um processo, interromper um cronômetro, (…)

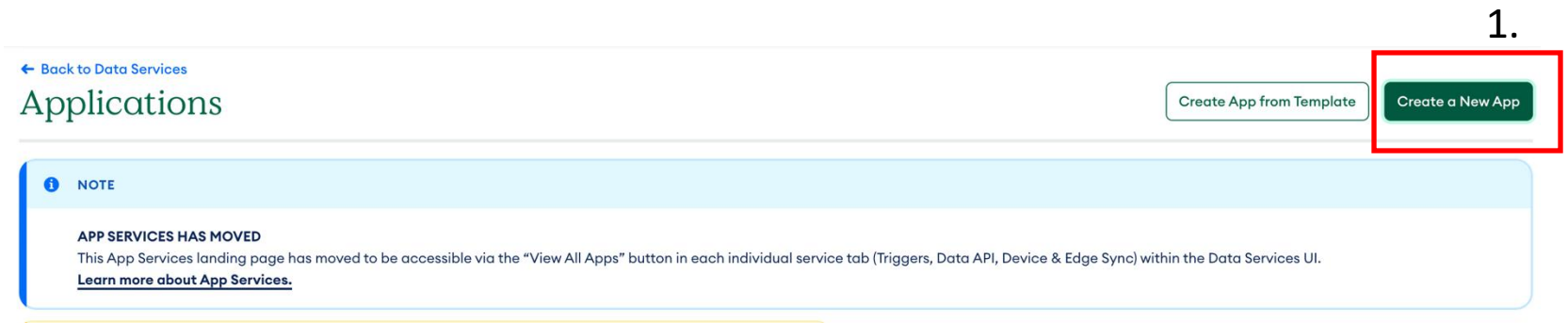
MongoDB Data API e HTTPS Endpoints Atlas

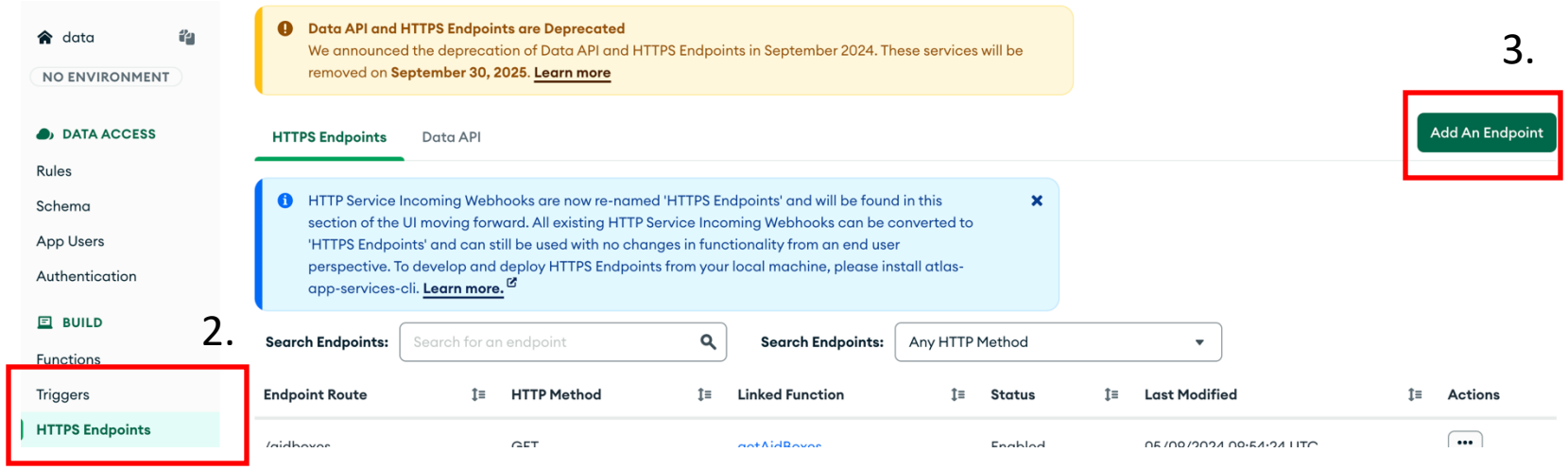
- São serviços intermediários totalmente geridos para interação com o MongoDB via HTTPS, sem a necessidade de um driver.
- MongoDB Data API e HTTPS Endpoints fornecem a conectividade simplificada e escalável.
- É utilizado JavaScript para desenvolver a lógica associada a cada endpoint.
- Recentemente foi anunciado que este serviço será desativado em Setembro de 2025.
- De qualquer forma, a lógica de implementação é independente do serviço e continua a ser uma abordagem prática e integradora para o desenvolvimento de APIs.

MongoDB Data API e HTTPS Endpoints



MongoDB Data API e HTTPS Endpoints





MongoDB Data API e HTTPS Endpoints

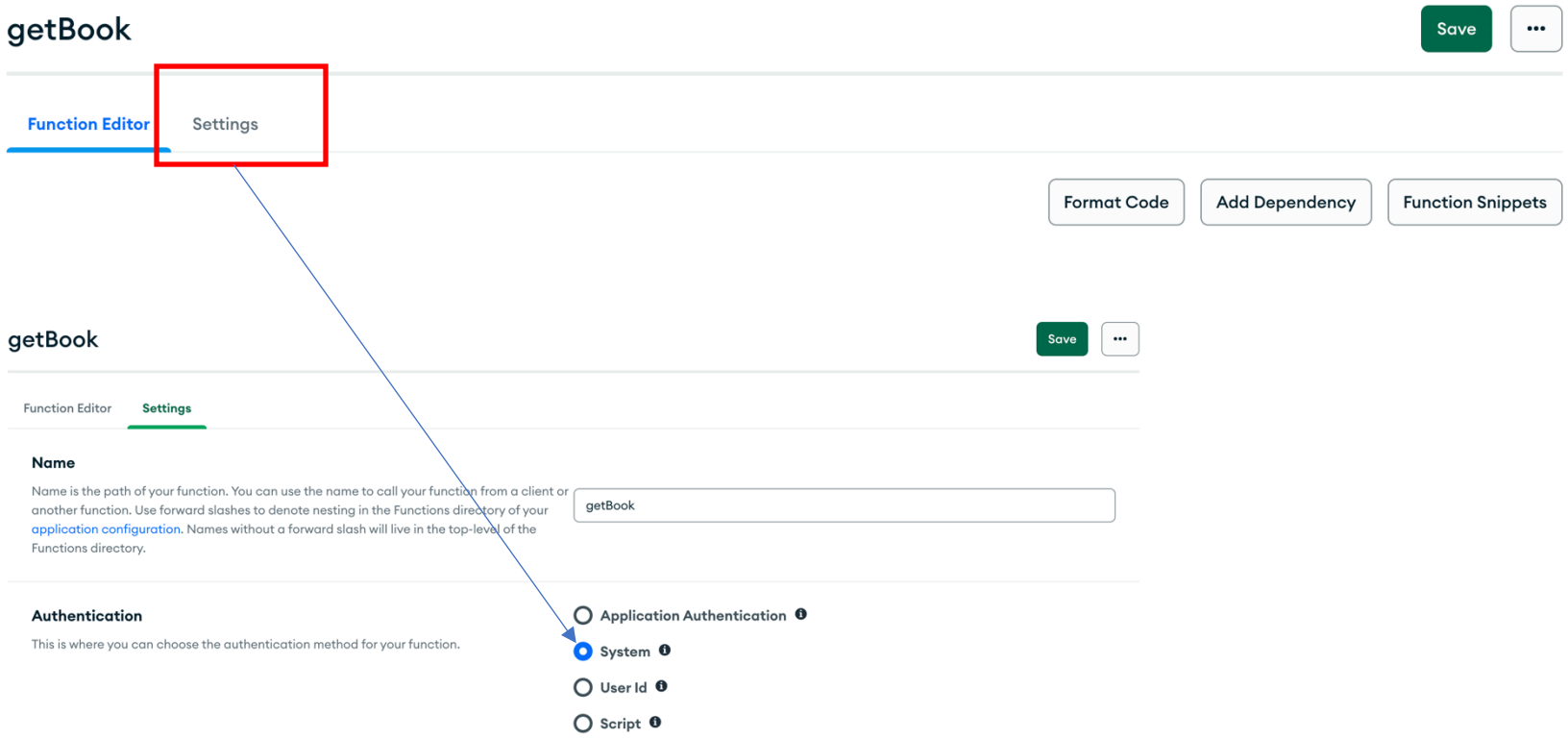
- A criação de um endpoint envolve:
  - A criação da Route. Por exemplo: /getBooks
  - O HTTP Method (GET, POST, ..)
  - Deve ser ativada a opção: Respond With Result
  - Para simplificar, em Authorization, colocar: No Additional Authorization
  - É necessário associar uma Function que terá a lógica (em javascript) associada ao Endpoint.
  - Os restantes parâmetros ficam por defeito.

MongoDB Data API e HTTPS Endpoints - GetBooks

```
// O endpoint será usado para interagir com uma base de dados no MongoDB Atlas.
exports = function({ query, headers, body }, response) {
        // A função é chamada quando o endpoint HTTP é acionado.
        // Os parâmetros disponíveis são:
        // - query: contém os parâmetros da query string da requisição.
        // - headers: contém os cabeçalhos HTTP enviados pelo cliente.
        // - body: contém o corpo da requisição (se aplicável).

        // Obtemos uma referência ao serviço MongoDB configurado no MongoDB Atlas App Services.
        // "Cluster0" é o nome do cluster de base de dados que configurámos.
        const cluster = context.services.get("Cluster0");

        // A partir do cluster, selecionamos a base de dados chamada "bookstore".
        const db = cluster.db("bookstore");

        // Selecionamos a coleção chamada "books" dentro da base de dados.
        const collection = db.collection("books");

        // Executamos uma consulta para obter todos os documentos da coleção "books".
        // O método "find()" retorna um cursor que contém os documentos da coleção.
        const doc = collection.find();

        // Devolvemos os documentos obtidos pela consulta.
        // O resultado será enviado como resposta para o cliente que fez a requisição HTTP.
        return doc;
};
```

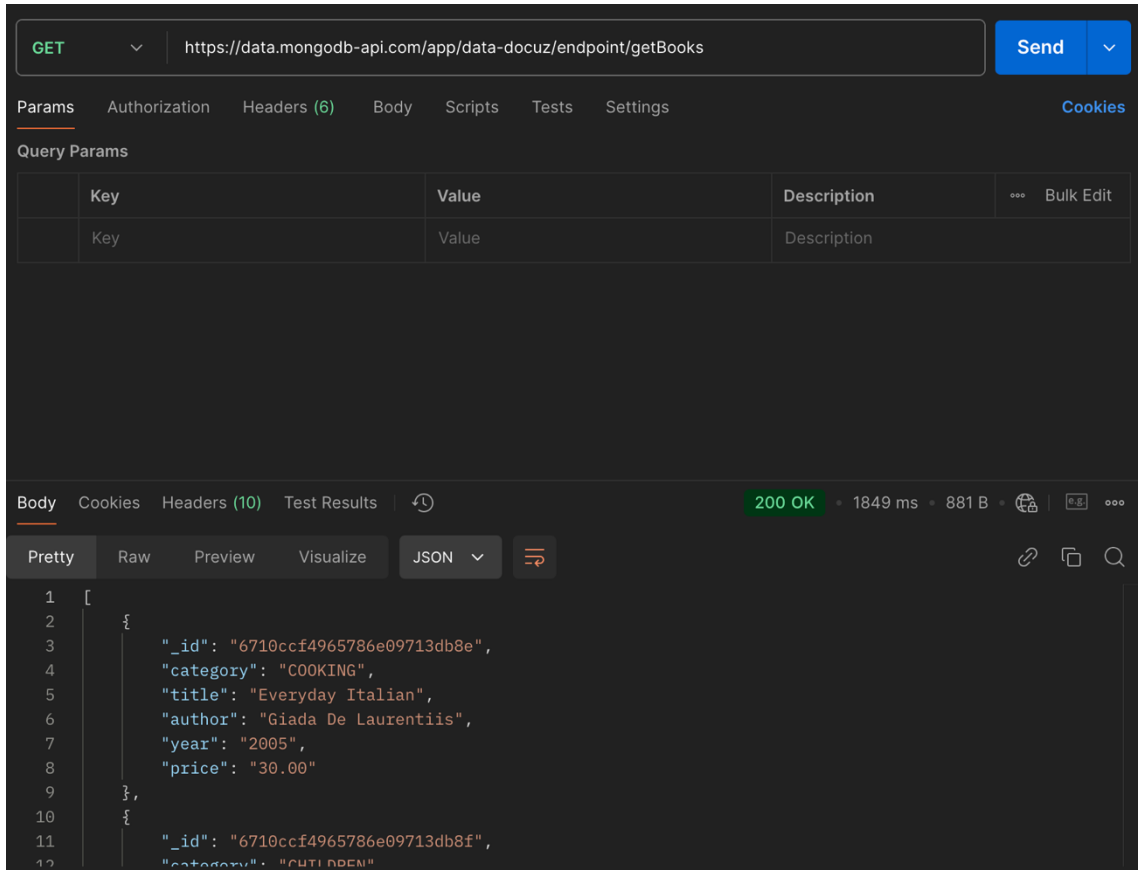
MongoDB Data API e HTTPS Endpoints

- Na Authentication (settings) assegurar que as funções a authorization está como "system":



MongoDB Data API e HTTPS Endpoints

- Testar utilizando (por exemplo) Postman:



MongoDB Data API e HTTPS Endpoints – GetBook by Title

```
exports = async function({ query }) {

try {
        // Obtemos o valor do parâmetro "title" da query string.
        const title = query.title;
        if (!title) {
                return { status: 400, error: "Parâmetro 'title' é obrigatório." };
        }

        const collection = context.services
        .get("Cluster0") // Nome do cluster configurado
        .db("bookstore") // Nome da base de dados
        .collection("books"); // Nome da coleção

        const book = await collection.findOne({ title: title });

        if (!book) {
        return { status: 404, error: `Nenhum livro encontrado com o título: '${title}'` };
        }

        return book;
        } catch (error) {
                return { status: 500, error: "Erro ao devolver o livro.", details: error.message };
        }
};
```

https://data.mongodb-api.com/app/data-docuz/endpoint/getBook?title=Everyday Italian

MongoDB Data API e HTTPS Endpoints – AddBook

```
exports = async function({ body }) {
    try {
        // Tentamos fazer o parsing do corpo da requisição para obter os dados do livro.
        const book = JSON.parse(body.text());
        if (!book.title || !book.author || !book.year || !book.category) {
            return {
                status: 400,
                error: "Os campos 'title', 'author', 'year' e 'genre' são obrigatórios."
                };
        }

        const collection = context.services
        .get("Cluster0") // Nome do cluster configurado
        .db("bookstore") // Nome da base de dados
        .collection("books"); // Nome da coleção

        const result = await collection.insertOne(book);
        book._id = result.insertedId;

        return book;
    } catch (error) {
        return {
            status: 500,
            error: "Erro ao adicionar o livro.",
            details: error.message
            };
    }
};
```

MongoDB Data API e HTTPS Endpoints – AddBook

- URL para adicionar o livro (POST):

  https://data.mongodb-api.com/app/data-docuz/endpoint/addBook

- Exemplo do Body:

```
{
    "category": "Fantasy",
    "title": "Game of Thrones",
    "author": "George R. R. Martin",
    "year": "2015",
    "price": "30.00"
}
```

MongoDB Data API e HTTPS Endpoints – UpdateBook

```
exports = async function({ query, body }) {
try {
        const title = query.title;

        // Validamos se o parâmetro "title" foi fornecido.
        if (!title) {
                return {
                status: 400,
                message: "Parâmetro 'title' é obrigatório na query string."
                };
    }

    const updateFields = JSON.parse(body.text());

        // Validamos se o corpo contém campos para atualização.
        if (!updateFields || Object.keys(updateFields).length === 0) {
                return {
                status: 400,
                message: "O corpo da requisição deve conter os campos para atualização."
                };
        }

        const collection = context.services.get("Cluster0").db("bookstore").collection("books");

        const result = await collection.updateOne(
                { title: title }, // Filtro para localizar o livro pelo título
                { $set: updateFields } // Atualizamos apenas os campos fornecidos no corpo
        );
```

MongoDB Data API e HTTPS Endpoints – UpdateBook

```
        // Verificamos se algum documento foi encontrado e atualizado.
        if (result.matchedCount === 0) {
                return {
                status: 404,
                message: `Nenhum livro encontrado com o título: '${title}'`
                };
        }

        return {
                status: 200,
                message: "Livro atualizado com sucesso.",
                updatedFields: updateFields
        };
        } catch (error) {
                return {
                status: 500,
                message: "Erro ao atualizar o livro.",
                details: error.message
                };
        }
};
```

MongoDB Data API e HTTPS Endpoints – UpdateBook

- URL para adicionar o livro (PATCH):

  https://data.mongodb-api.com/app/data-docuz/endpoint/updateBook?title=Everyday Italian

- Exemplo do Body:

```
{
        "price": "50.00"
}
```

MongoDB Data API e HTTPS Endpoints – ReplaceBook

```
exports = async function({ query, body }) {
        try {
        const title = query.title;

        if (!title) {
        return {
        status: 400,
        message: "Parâmetro 'title' é obrigatório na query string."
        };
        }
        const newBook = JSON.parse(body.text());

        if (!newBook.author || !newBook.year || !newBook.category) {
                return {
                status: 400,
                message: "Os campos 'author', 'year' e 'genre' são obrigatórios no corpo da requisição."
                };
        }

        newBook.title = title;

        const collection = context.services.get("Cluster0").db("bookstore").collection("books");
```

MongoDB Data API e HTTPS Endpoints – ReplaceBook

```
        // Realiza a operação de substituição (ou criação se o título não existir).
        const result = await collection.replaceOne(
                { title: title }, // Filtro para localizar o livro pelo título
                newBook, // Substitui completamente pelo novo livro
                { upsert: true } // Cria um novo recurso se não existir
        );

        // Verifica se o recurso foi criado ou atualizado.
        const message = result.upsertedId? "Livro criado com sucesso.": "Livro atualizado com sucesso.";

        // Retorna o novo estado do livro.
        return newBook;

        } catch (error) {
        // Tratamos erros que possam ocorrer durante a operação.
                return {
                status: 500,
                message: "Erro ao processar a requisição.",
                details: error.message
                };
        }
};
```

MongoDB Data API e HTTPS Endpoints – ReplaceBook

- URL para adicionar o livro (PUT):

  https://data.mongodb-api.com/app/data-docuz/endpoint/replaceBook?title=Everyday Italian

- Exemplo do Body:

```
{
"category": "Italian COOKING",
"author": "Giada De Laurentiis 2",
"year": "2010",
"price": "35.00"
}
```

MongoDB Data API e HTTPS Endpoints – DeleteBook

```
exports = async function({ query }) {
    try {
        const title = query.title;

        if (!title) {
            return {
                status: 400,
                message: "Parâmetro 'title' é obrigatório na query string."
            };
        }

        const collection = context.services.get("Cluster0").db("bookstore").collection("books");

        const result = await collection.deleteOne({ title: title });

        if (result.deletedCount === 0) {
            return {
                status: 404,
                message: `Nenhum livro encontrado com o título: '${title}'`
            };
        }

        return {
            status: 200,
            message: `Livro '${title}' eliminado com sucesso.`
        };
    }
```

MongoDB Data API e HTTPS Endpoints – DeleteBook

```
catch (error) {
        return {
                status: 500,
                message: "Erro ao deletar o livro.",
                details: error.message
                };
        }
};
```

MongoDB Data API e HTTPS Endpoints – DeleteBook

- URL para adicionar o livro (DELETE):

  https://data.mongodb-api.com/app/data-docuz/endpoint/deleteBook?title=Everyday Italian 2

Referências

- Referências Web:
  - https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Basico_sobre_HTTP
  - https://www.restapitutorial.com/
  - https://www.postman.com
- Livro:
  - Lauret A., The Design of Web APIs, Manning, 2019;

P.PORTO

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Introdução Web APIs

Bruno Oliveira (bmo@estg.ipp.pt)

PEI Processamento Estruturado de Informação