

Projeto 02 - Blog Pessoal - Spring Security 03

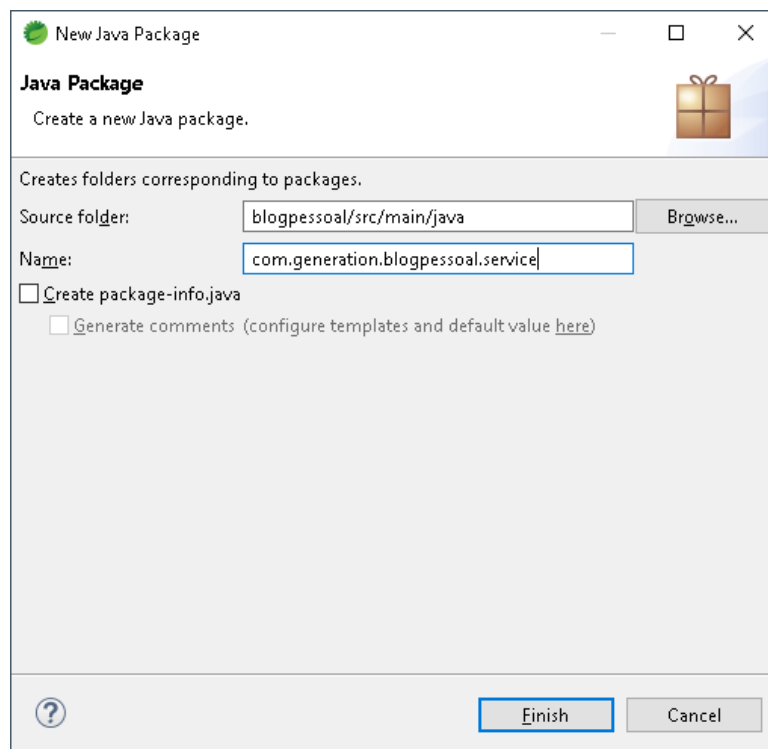
O que veremos por aqui:

1. A Classe UsuarioService
2. A Classe UsuarioController
3. Atualização da Classe PostagemController
4. Testes no Postman

Vamos finalizar o Ecossistema do Usuário.

Passo 01 - Criar o Pacote Service

1. Na Guia **Package explorer**, clique com o botão direito do mouse sobre a Package **com.generation.blogpessoal**, na Source Folder **src/main/java** e clique na opção **New → Package**.
2. Na janela **New Java Package**, no item **Name**, acrescente no final do nome da Package **.service** e clique no botão **Finish** para concluir.



Passo 02 - Criar a Classe UsuarioService na Camada Service

A **Classe UsuarioService** é responsável por manipular as regras de negócio de usuário no sistema. Esta classe deve ser anotada com a anotação **@Service**, para que o Spring identifique que é uma classe que serviço e carregue ela sempre que puder. Vale mencionar que alguns métodos definidos abaixo são de extrema importância para o sistema.

1. Clique com o botão direito do mouse sobre o **Pacote Security (com.generation.blogpessoal.service)**, na Source Folder Principal (**src/main/java**), e clique na opção **New → Class**
2. Na janela **New Java Class**, no item **Name**, digite o nome da Classe (**UsuarioService**), e na sequência clique no botão **Finish** para concluir.

A seguir veja sua implementação:

```
package com.generation.blogpessoal.service;

import java.nio.charset.Charset;
import java.util.Optional;

import org.apache.commons.codec.binary.Base64;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.web.server.ResponseStatusException;

import com.generation.blogpessoal.model.Usuario;
import com.generation.blogpessoal.model.UsuarioLogin;
import com.generation.blogpessoal.repository.UsuarioRepository;

@Service
public class UsuarioService {

    @Autowired
    private UsuarioRepository usuarioRepository;

    public Optional<Usuario> cadastrarUsuario(Usuario usuario) {

        if (usuarioRepository.findByUsuario(usuario.getUsuario())
            .isPresent())
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST,
                "Usuário já existe!", null);

        usuario.setSenha(criptografarSenha(usuario.getSenha()));

        return Optional.of(usuarioRepository.save(usuario));
    }

    public Optional<Usuario> atualizarUsuario(Usuario usuario) {

        if (usuarioRepository.findById(usuario.getId()).isPresent()) {
            Optional<Usuario> buscaUsuario = usuarioRepository.
                findByUsuario(usuario.getUsuario());

            if (buscaUsuario.isPresent()) {
```

```

        if (buscaUsuario.get().getId() != usuario.getId())
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST,
                "Usuário já existe!", null);
    }

    usuario.setSenha(criptografarSenha(usuario.getSenha()));

    return Optional.of(usuarioRepository.save(usuario));
}

throw new ResponseStatusException(HttpStatus.NOT_FOUND,
    "Usuário não encontrado!", null);
}

public Optional<UsuarioLogin> logarUsuario(
    Optional<UsuarioLogin> usuarioLogin) {

    Optional<Usuario> usuario = usuarioRepository
        .findByUsuario(usuarioLogin.get().getUsuario());

    if (usuario.isPresent()) {
        if (compararSenhas(usuarioLogin.get().getSenha(),
            usuario.get().getSenha())) {

            usuarioLogin.get().setId(usuario.get().getId());
            usuarioLogin.get().setNome(usuario.get().getNome());
            usuarioLogin.get().setFoto(usuario.get().getFoto());
            usuarioLogin.get().setToken(
                gerarBasicToken(usuarioLogin.get().getUsuario(),
                    usuarioLogin.get().getSenha()));
            usuarioLogin.get().setSenha(usuario.get().getSenha());

            return usuarioLogin;
        }
    }

    throw new ResponseStatusException(
        HttpStatus.UNAUTHORIZED, "Usuário ou senha inválidos!", null);
}

private String criptografarSenha(String senha) {

    BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
    String senhaEncoder = encoder.encode(senha);

    return senhaEncoder;
}

private boolean compararSenhas(String senhaDigitada,
    String senhaBanco) {
    BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();

    return encoder.matches(senhaDigitada, senhaBanco);
}

private String gerarBasicToken(String email, String password) {
    String estrutura = email + ":" + password;

```




```

        byte[] estruturaBase64 = Base64.encodeBase64(
            estrutura.getBytes(Charset.forName("US-ASCII")));
        return "Basic " + new String(estruturaBase64);
    }
}

```

Observe que foram criados os Métodos `cadastrarUsuario()` e `atualizarUsuario()`, que criptografam a senha e impedem a duplicação do usuário no Banco de dados. O Método `logarUsuario()`, além de autenticar o usuário no sistema, ele compara a senha digitada pelo usuário com a senha persistida no Banco de dados e gera o Token do usuário.

Para executar as operações: Criptografar Senha, Comparar Senhas e GerarBasicToken foram criados os respectivos Métodos auxiliares na própria Classe `UsuarioService`.

	ALERTA DE BSM: Mantenha a Atenção aos Detalhes, o cadastro de um novo usuário no sistema necessita ser validado no Banco de dados. Caso o usuário já exista, a aplicação não deve permitir que ele seja criado novamente, pois um usuário duplicado no sistema ocasionará um erro HTTP Status 500.
	ALERTA DE BSM: Mantenha a Atenção aos Detalhes, ao atualizar um usuário é importante que seja validado novamente a criptografia da senha e o usuário (e-mail). Caso não seja validado ocasionará um problema ao tentar pegar as credenciais pelo front-end da aplicação.
	ALERTA DE BSM: Mantenha a Atenção aos Detalhes, ao utilizar o método para <code>logarUsuario</code>, se atentar de que seja passado todos os parâmetros do <code>usuarioLogin</code>, pois o mesmo será utilizado pelo front-end da aplicação.

Para concluir, não esqueça de Salvar o código (**File → Save All**).



[Documentação: @Service](#)



[Documentação: BCryptPasswordEncoder - Java Doc Spring](#)



[Documentação: Throw](#)



[Documentação: Métodos de Referência \(::\)](#)



[Documentação: Base64 - Java Doc Commons](#)



[Código fonte: Classe UsuarioService](#)



Passo 03 - Criar a Classe `UsuarioController` na Camada Controller

A classe **UsuarioController**, é responsável por fornecer o acesso aos recursos do sistema. Uma das suas funcionalidades abaixo é promover o CRUD do usuário. Além de permitir total manipulação do usuário, consumindo os serviços cadastrar usuário, atualizar usuário e autenticar da Classe UsuarioService.

1. Clique com o botão direito do mouse sobre o **Pacote Security (com.generation.blogpessoal.controller)**, na Source Folder Principal (**src/main/java**), e clique na opção **New → Class**
2. Na janela **New Java Class**, no item **Name**, digite o nome da Classe (**UsuarioController**), e na sequência clique no botão **Finish** para concluir.

A seguir veja sua implementação:

```
package com.generation.blogpessoal.controller;

import java.util.List;
import java.util.Optional;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.generation.blogpessoal.model.Usuario;
import com.generation.blogpessoal.model.UsuarioLogin;
import com.generation.blogpessoal.repository.UsuarioRepository;
import com.generation.blogpessoal.service.UsuarioService;

@RestController
@RequestMapping("/usuarios")
@CrossOrigin(origins = "*", allowedHeaders = "*")
public class UsuarioController {

    @Autowired
    private UsuarioService service;

    @Autowired
    private UsuarioRepository repository;

    @GetMapping("/all")
    public ResponseEntity <List<Usuario>> getAll() {
        return ResponseEntity.ok(repository.findAll());
    }

    @GetMapping("/{id}")
    public ResponseEntity<Usuario> getById(@PathVariable long id) {
        return repository.findById(id)
            .map(resp -> ResponseEntity.ok(resp))
    }
}
```

```

        .orElse(ResponseEntity.notFound().build());
    }

    @PostMapping("/logar")
    public ResponseEntity<UsuarioLogin> authenticationUsuario(
        @RequestBody Optional<UsuarioLogin> usuario) {
        return service.logarUsuario(usuario)
            .map(resp -> ResponseEntity.ok(resp))
            .orElse(ResponseEntity.status(HttpStatus.UNAUTHORIZED).build());
    }

    @PostMapping("/cadastrar")
    public ResponseEntity<Usuario> postUsuario(
        @Valid @RequestBody Usuario usuario) {
        return service.cadastrarUsuario(usuario)
            .map(resp -> ResponseEntity.status(HttpStatus.CREATED).body(resp))
            .orElse(ResponseEntity.status(HttpStatus.BAD_REQUEST).build());
    }

    @PutMapping("/atualizar")
    public ResponseEntity<Usuario> putUsuario(
        @Valid @RequestBody Usuario usuario){
        return service.atualizarUsuario(usuario)
            .map(resp -> ResponseEntity.status(HttpStatus.OK).body(resp))
            .orElse(ResponseEntity.status(HttpStatus.NOT_FOUND).build());
    }
}

```

Observe que nos Métodos Cadastrar, Atualizar e Autenticar, ao invés de usarmos a injeção de dependência da Interface `UsuarioRepository`, estamos utilizando a injeção de dependência da Classe de Serviço `UsuarioService` porque os 3 Métodos foram implementados nela.



ALERTA DE BSM: Mantenha a Atenção aos Detalhes, nos métodos `postUsuario` e `putUsuario` para não esquecer a notação `@Valid`. Caso ela não seja inserida, ao fornecer parâmetros inválidos, o servidor retornará o HTTP Status 500, ao invés do HTTP Status 400.

Para concluir, não esqueça de Salvar o código (**File → Save All**).



[Código fonte: Classe `UsuarioController`](#)



Passo 04 - Testar o Recurso Usuario no Insomnia

Vamos criar no Insomnia todas as requisições necessárias para testar os 5 Métodos do Recurso Usuario. Veja abaixo como ficam as requisições para testar o Recurso Usuario:



DICA: Caso você tenha alguma dúvida sobre como criar as Requisições, consulte a [Documentação dos Recursos Postagem e Tema](#).

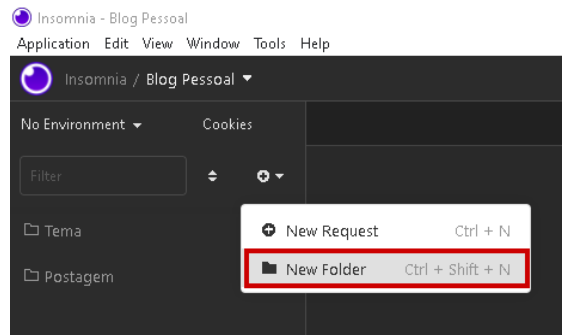


ATENÇÃO: Depois de criar o Relacionamento entre Classes, todas as Consultas dos Recursos Postagem, Tema e Usuario trarão os Objetos associados.

4.1. Criando a Pasta Usuario

Vamos criar dentro da **Collection Blog Pessoal** a **Pasta Usuario**, que guardará todas as requisições do **Recurso Usuario**.

1. Na **Collection Blog Pessoal**, clique no botão . No menu que será aberto, clique na opção **New Folder**.



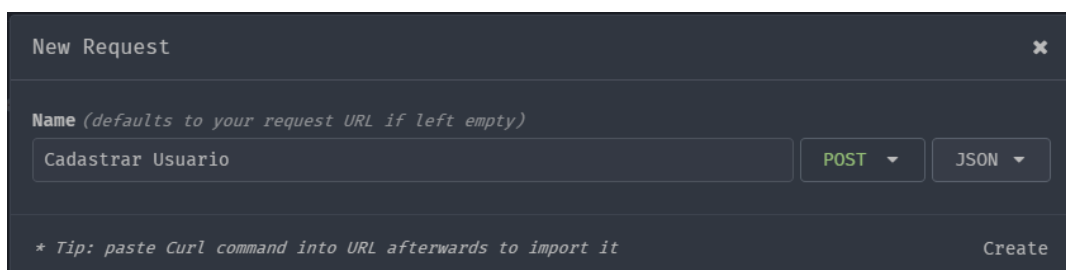
2. Na janela que será aberta, informe o nome da pasta (**Usuario**) e clique no botão **Create** para concluir.



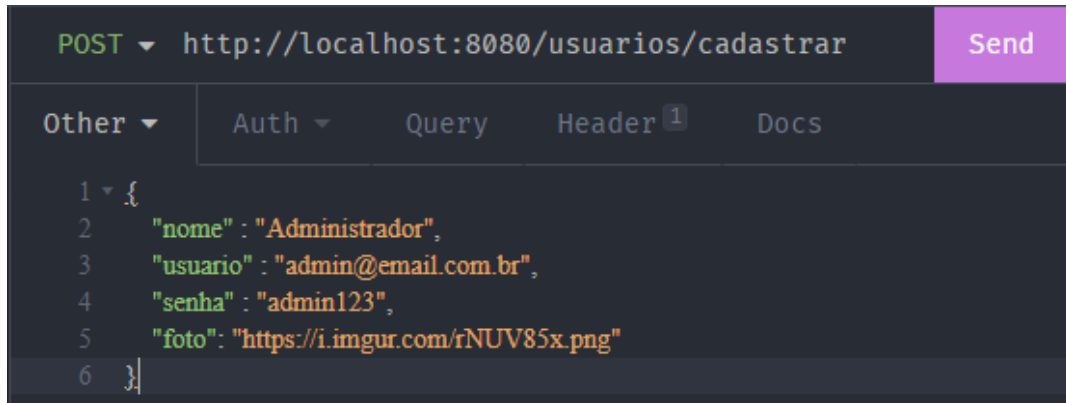
4.2. Criando a Request - Cadastrar Usuario

Vamos começar pela requisição Cadastrar Usuario porque sem um usuário cadastrado não será possível autenticar (logar) no sistema e acessar os demais endpoints.

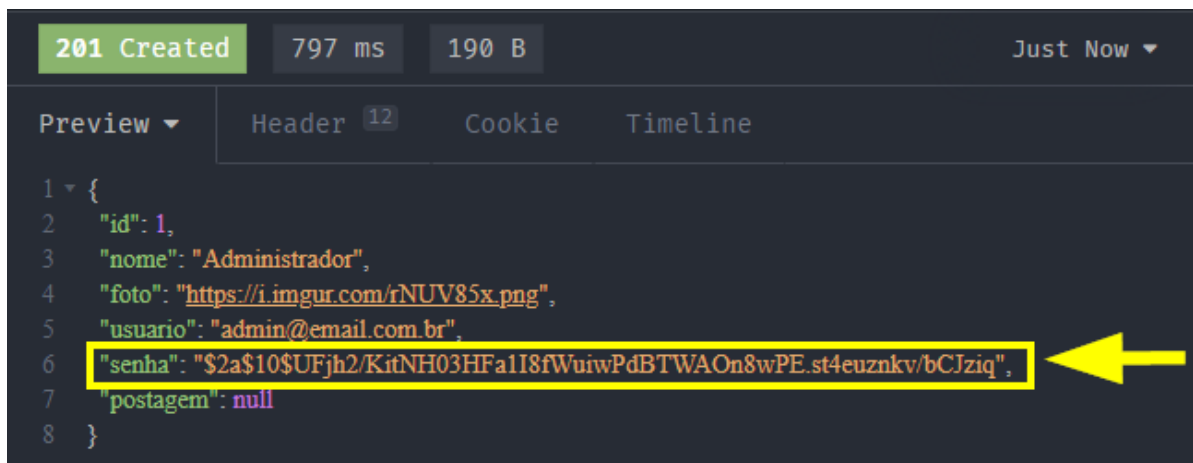
1. Clique com o botão direito do mouse sobre a **Pasta Usuario** para abrir o menu e clique na opção **New Request**.
2. Na janela que será aberta, informe o nome da requisição e o Método HTTP que será utilizado (**POST**). Depois de selecionar o Método **POST**, observe que será necessário informar o formato em que os dados serão enviados através do Corpo da Requisição. Selecione o formato **JSON** e clique no botão **Create** para concluir.



3. Configure a requisição conforme a imagem abaixo:



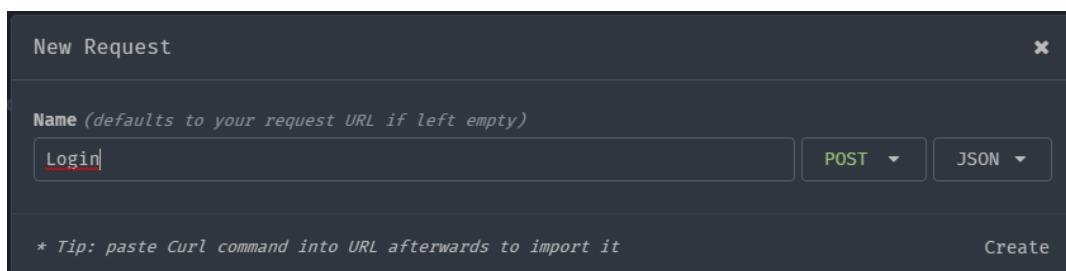
4. Observe que na requisição do tipo Post o Corpo da requisição (Request Body), deve ser preenchido com um JSON contendo o nome, o usuario(e-mail), a senha e a foto(link), que vc deseja persistir no Banco de dados.
5. Observe que depois de persistir os dados do usuario, a aplicação retornará a senha criptografada.



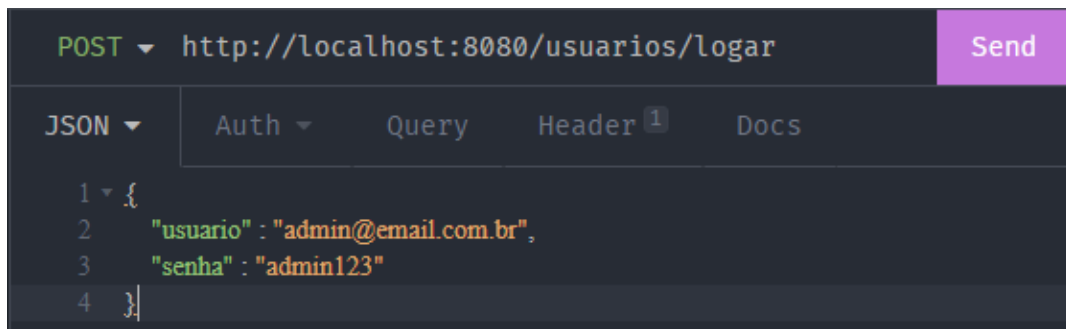
4.3. Criando a Request - Autenticar Usuario (Login)

Usuário persistido, agora vamos efetuar o login no sistema, que nos retornará o **Token de Autorização**, que nos permitirá consumir os demais endpoints e recursos da aplicação.

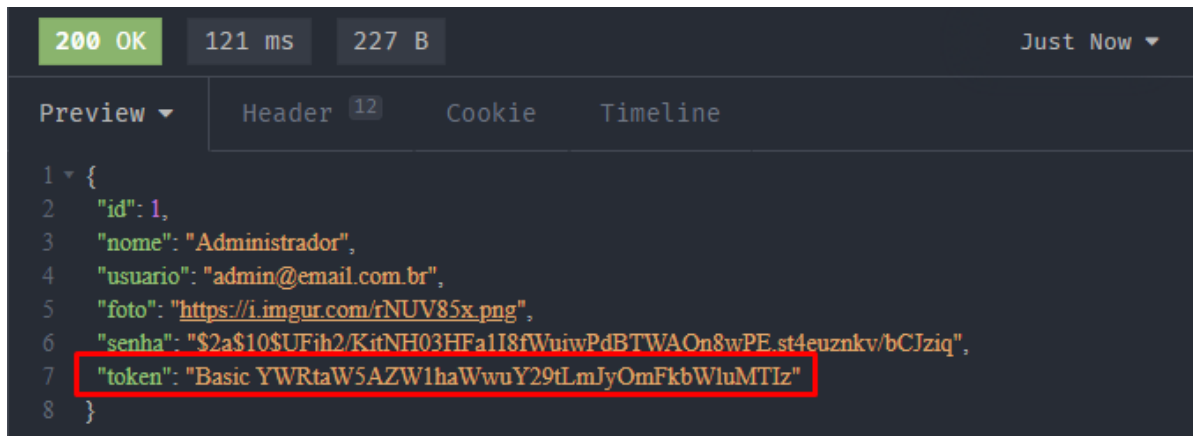
1. Clique com o botão direito do mouse sobre a **Pasta Usuario** para abrir o menu e clique na opção **New Request**.
2. Na janela que será aberta, informe o nome da requisição e o Método HTTP que será utilizado (**POST**). Depois de selecionar o Método **POST**, observe que será necessário informar o formato em que os dados serão enviados através do Corpo da Requisição. Selecione o formato **JSON** e clique no botão **Create** para concluir.



3. Configure a requisição conforme a imagem abaixo:

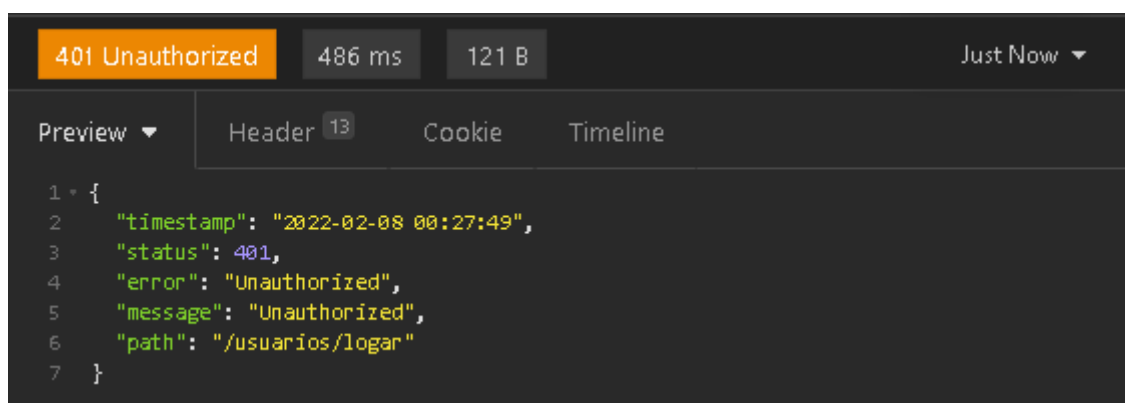


4. Observe que no JSON estamos passando apenas o usuário e a senha, porque são as únicas informações que utilizaremos no login. A Classe UsuarioLogin possui outros atributos que serão preenchidos pela aplicação (se o login for efetuado com sucesso) e retornados no JSON da resposta, como vemos na figura abaixo:



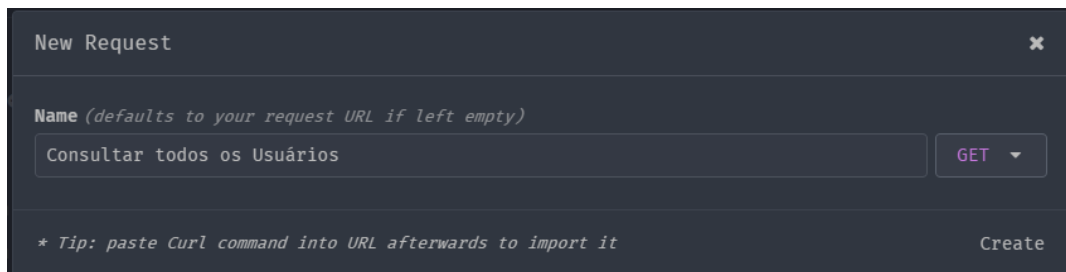
5. Observe que o Token foi gerado e enviado na resposta. Copie o Token da resposta porque vamos precisar dele nas próximas requisições.

6. Caso o login falhe, você receberá o **status 401 (Unauthorized)**, como mostra a figura abaixo:



4.4. Criando a Request - Consultar todos os Usuarios - findAll()

1. Clique com o botão direito do mouse sobre a **Pasta Usuario** para abrir o menu e clique na opção **New Request**.
2. Na janela que será aberta, informe o nome da requisição e o Método HTTP que será utilizado (**GET**). Clique no botão **Create** para concluir.



New Request

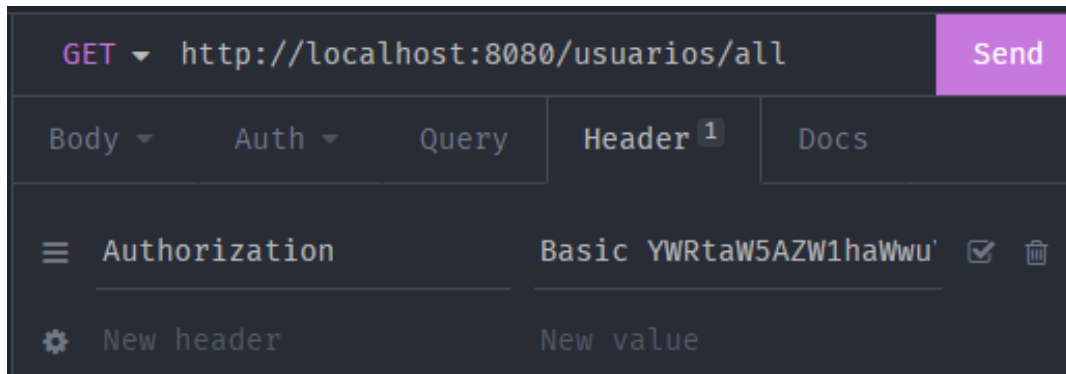
Name (defaults to your request URL if left empty)

Consultar todos os Usuários GET

* Tip: paste Curl command into URL afterwards to import it

Create

3. Configure a requisição conforme a imagem abaixo:



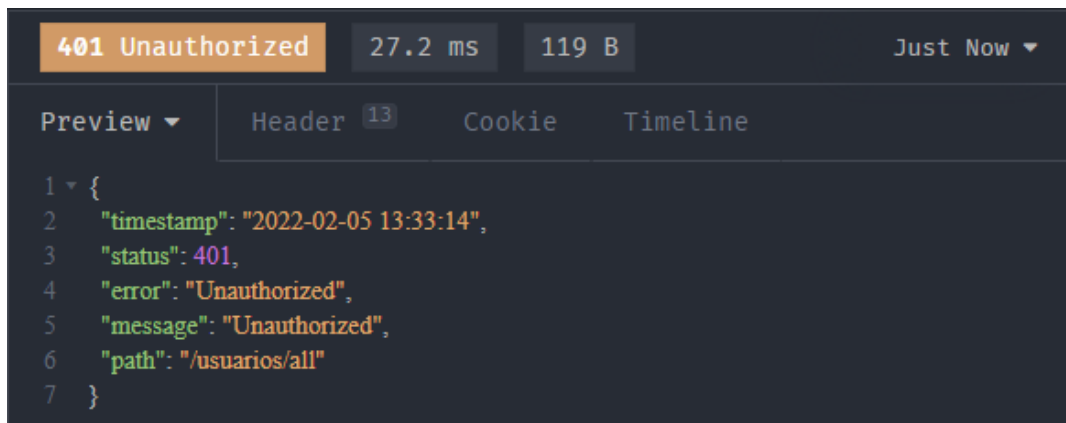
GET http://localhost:8080/usuarios/all Send

Body Auth Query Header 1 Docs

Authorization Basic YWRtaW5AZW1haWwu' [check] [trash]

New header New value

4. Como a segurança foi habilitada, caso você envie a requisição do jeito que está, você receberá o **status 401 (Unauthorized)**, como mostra a figura abaixo:



401 Unauthorized 27.2 ms 119 B Just Now

Preview Header 13 Cookie Timeline

```
1 {
2   "timestamp": "2022-02-05 13:33:14",
3   "status": 401,
4   "error": "Unauthorized",
5   "message": "Unauthorized",
6   "path": "/usuarios/all"
7 }
```

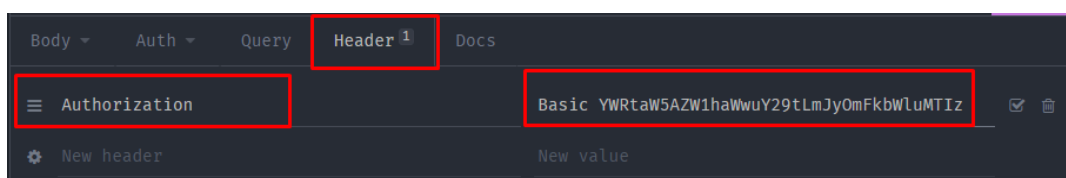
5. A explicação é simples: **Você precisa passar no cabeçalho da requisição o Token recebido no login**, caso contrário não será possível efetuar nenhuma operação dentro da API.



ATENÇÃO: No Frontend, independente de ser implementado no Angular ou no React, a passagem do Token será automatizada.

6. Clique na Guia **Header** e adicione uma nova linha de cabeçalho.

7. Na coluna **New header** digite **Authorization**. Na coluna **New value** da linha insira o **Token** que você recebeu no Login, como mostra a figura abaixo:



Body Auth Query Header 1 Docs

Authorization Basic YWRtaW5AZW1haWwuY29tLmJyOmFkbWluMTIz [check] [trash]

New header New value

8. Agora a sua requisição funcionará!



ALERTA DE BSM: Mantenha a Atenção aos Detalhes! Você deverá inserir o token no cabeçalho de todas requisições dos Recursos Postagem, Tema e Usuario (Exceto Cadastrar e Logar), caso contrário você receberá o status 401 (Unauthorized) em todas elas.

4.5. Criando a Request - Consultar Usuario por ID - findByld(id)

1. Clique com o botão direito do mouse sobre a **Pasta Usuario** para abrir o menu e clique na opção **New Request**.
2. Na janela que será aberta, informe o nome da requisição e o Método HTTP que será utilizado (**GET**). Clique no botão **Create** para concluir.

New Request

Name (defaults to your request URL if left empty)

Consultar Usuário por ID GET

* Tip: paste Curl command into URL afterwards to import it

Create

3. Configure a requisição conforme a imagem abaixo:

GET http://localhost:8080/usuarios/1 Send

Body Auth Query Header 1 Docs

Authorization Basic YWRtaW5AZW1haWwu

New header New value

4. Clique na Guia **Header** e adicione uma nova linha de cabeçalho.
5. Na coluna **New header** digite **Authorization**. Na coluna **New value** da linha insira o **Token** que você recebeu no Login, como mostra a figura abaixo:

Body Auth Query Header 1 Docs

Authorization Basic YWRtaW5AZW1haWwuY29tLmJyOmFkbWluMTIz

New header New value

4.6. Criando a Request - Atualizar Usuário

1. Clique com o botão direito do mouse sobre a **Pasta Usuario** para abrir o menu e clique na opção **New Request**.
2. Na janela que será aberta, informe o nome da requisição e o Método HTTP que será utilizado (**PUT**). Depois de selecionar o Método **PUT**, observe que será necessário informar o formato em que os dados serão enviados através do Corpo da Requisição. Selecione o formato **JSON** e clique no botão **Create** para concluir.

New Request

Name (defaults to your request URL if left empty)

Atualizar Usuário

PUT JSON

* Tip: paste Curl command into URL afterwards to import it

Create

3. Configure a requisição conforme a imagem abaixo:

PUT http://localhost:8080/usuarios/atualizar Send

Other Auth Query Header 2 Docs

```

1 {
2   "id" : 1,
3   "nome" : "Administrador Generation",
4   "usuario" : "admin@email.com.br",
5   "senha" : "admin123"
6 }

```

4. Observe que na requisição do tipo PUT o Corpo da requisição (Request Body), deve ser preenchido com um JSON contendo o Id, o nome, o usuario(e-mail), a senha e a foto(link) que vc deseja atualizar no Banco de dados.

5. Clique na Guia **Header** e adicione uma nova linha de cabeçalho.

6. Na coluna **New header** digite **Authorization**. Na coluna **New value** insira o **Token** que você recebeu no Login, como mostra a figura abaixo:

Body Auth Query Header 1 Docs

New header	New value
Authorization	Basic YWRtaW5AZW1haWwY29tLmJyOmFkbWluMTIz

Passo 05 - Atualizar as Requisições Cadastrar e Atualizar Postagem no Insomnia

Como habilitamos o Relacionamento entre as Classes Postagem e Usuario, para Cadastrarmos e Alterarmos as Postagens vamos precisar atender alguns requisitos:

- O Tema e o Usuario devem ser persistidos antes de criar a nova Postagem.
- Na requisição Cadastrar e Atualizar Postagem, o JSON enviado no Corpo da Requisição deve conter um Objeto da Classe Tema identificado apenas pelo **Atributo id** e um Objeto da Classe Usuario identificado apenas pelo **Atributo id**.

5.1. Atualização - Requisição Cadastrar Postagem

Vamos alterar o Corpo da requisição (Body), conforme a imagem abaixo:

```
POST http://localhost:8080/postagens Send

Other Auth Query Header 2 Docs

1 {
2   "titulo": "Postagem 01",
3   "texto": "Texto da Postagem 01",
4   "tema": {
5     "id": 1
6   },
7   "usuario": {
8     "id": 1
9   }
10 }
```

Observe que está sendo passado dentro do JSON um **Objeto da Classe Usuario** chamado **usuario**, identificado apenas pelo **Atributo id**.

Não esqueça de inserir o token no cabeçalho da requisição.



ATENÇÃO: O Objeto Usuario deve ser persistido no Banco de dados antes de ser inserido no JSON da requisição Cadastrar Postagem.

5.2. Atualização - Requisição Atualizar Postagem

Vamos alterar o Corpo da requisição (Body), conforme a imagem abaixo:

```
PUT http://localhost:8080/postagens/ Send

Other Auth Query Header 2 Docs

1 {
2   "id": "1",
3   "titulo": "Minha Postagem 12 - Atualizada!",
4   "texto": "Texto da postagem 12 atualizado!",
5   "tema": {
6     "id": 1
7   },
8   "usuario": {
9     "id": 1
10  }
11 }
```

Observe que está sendo passado dentro do JSON um **Objeto da Classe Usuario** chamado **usuario**, identificado apenas pelo **Atributo id**.

Não esqueça de inserir o token no cabeçalho da requisição.



ATENÇÃO: *O Objeto Usuario deve ser persistido no Banco de dados antes de ser inserido no JSON da requisição Atualizar Postagem.*



[Código fonte do projeto](#)