

Projeto 02 - Blog Pessoal - Spring Security 03

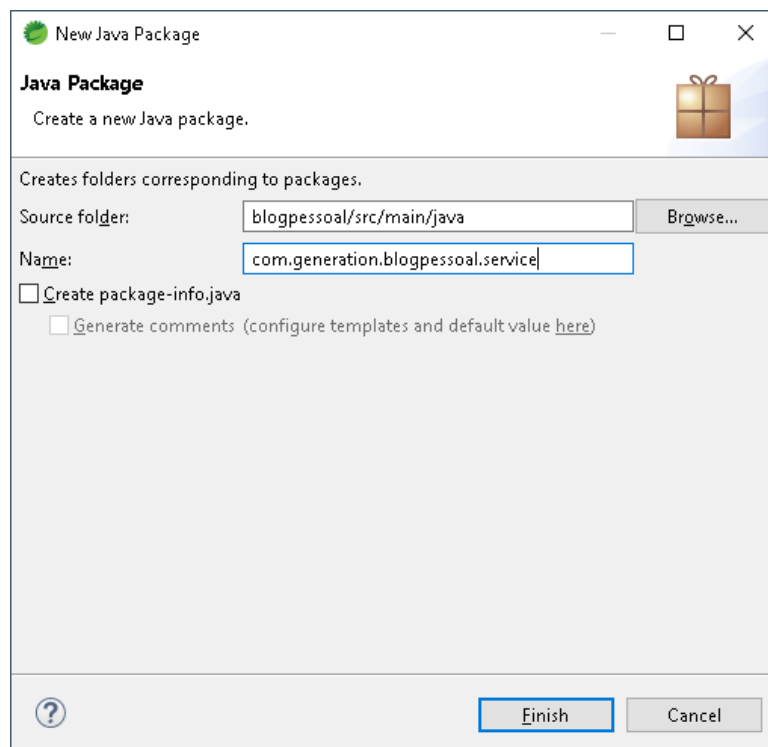
O que veremos por aqui:

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

Vamos finalizar o Ecosistema 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 logarUsuario, se atentar de que seja passado todos os parâmetros do usuarioLogin, 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 Postman

Vamos criar no Postman 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

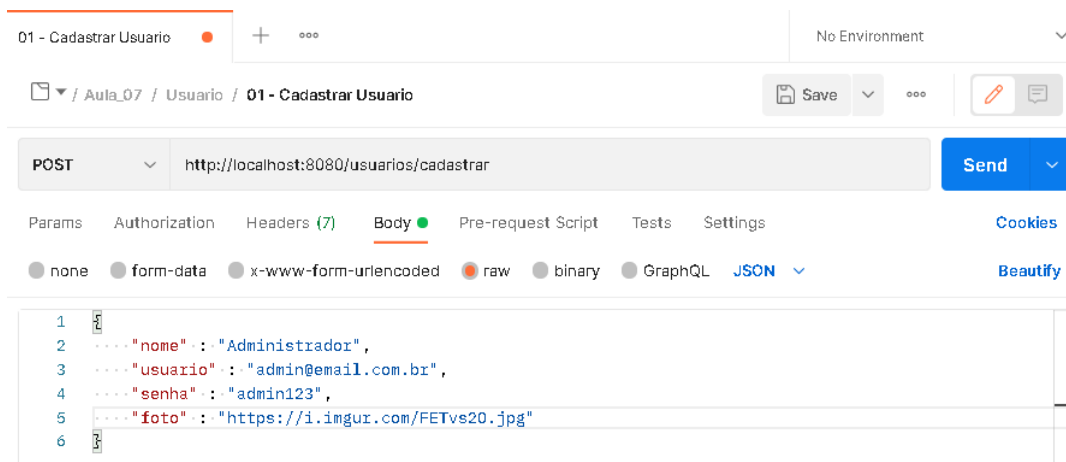
Vamos criar dentro da **Collection Blog Pessoal** a **Pasta Usuario**, que guardará todas as requisições do **Recurso Usuario**.

1. Na **Collection Blog Pessoal**, clique nos 3 pontinhos ao lado do nome da Collection, para abrir o menu e clique na opção **Add Folder**.
2. Na janela que será aberta, informe o nome da pasta (Usuario) e pressione a tecla (Enter) para concluir.

4.2. Criando a Request - Cadastrar Usuario

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

1. Na **Pasta Usuario**, clique nos 3 pontinhos ao lado do nome da pasta, para abrir o menu e clique na opção **Add Request**.
2. Configure a requisição conforme a imagem abaixo:



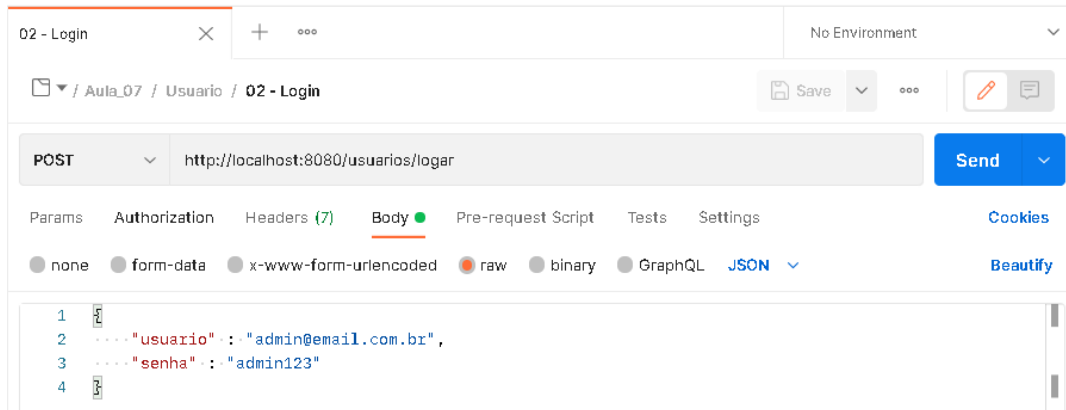
3. Observe que na requisição do tipo Post o Corpo da requisição (Request Body), deve ser preenchido com um JSON contendo o nome, o usuario(e-mail), a senha e a foto(link), que vc deseja persistir no Banco de dados.
4. Observe que depois de persistir os dados do usuario, a aplicação retornará a senha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



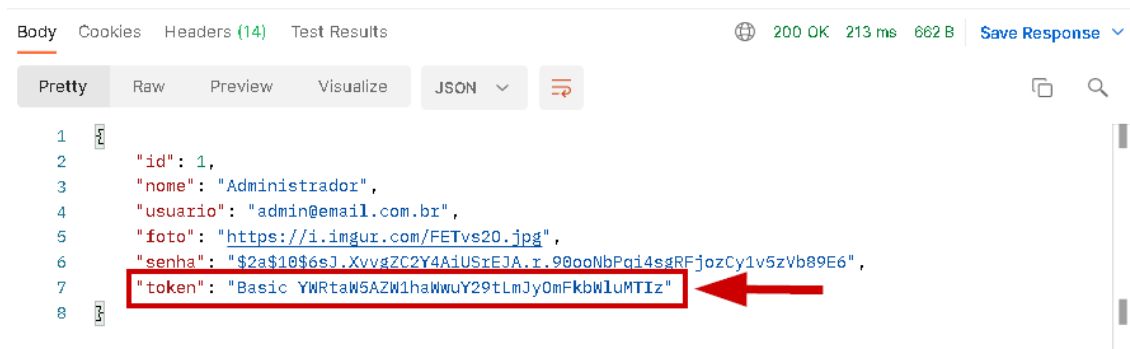
4.3. Criando a Request - Autenticar Usuario (Logar)

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. Na **Pasta Usuario**, clique nos 3 pontinhos ao lado do nome da pasta, para abrir o menu e clique na opção **Add Request**.
2. Configure a requisição conforme a imagem abaixo:



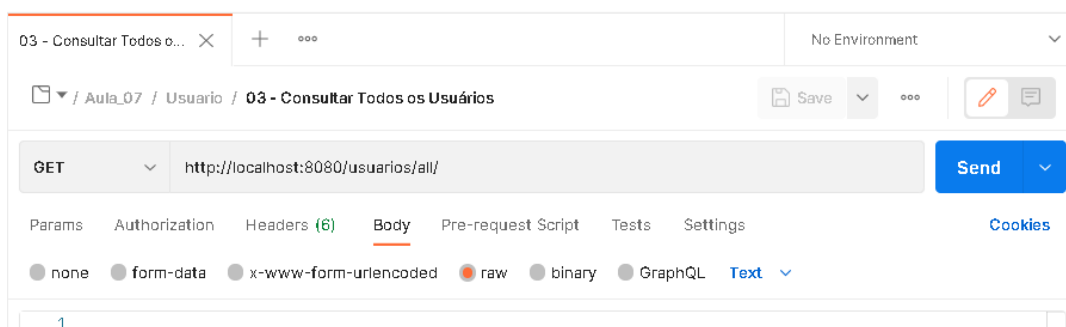
3. Observe que no JSON estamos passando apenas o usuário e a senha, porquê são as únicas informações que utilizaremos no login. A Classe `UsuarioLogin` possui outros atributos que serão preenchidos pela aplicação (se o login for efetuado com sucesso) e retornados no JSON da resposta, como vemos na figura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



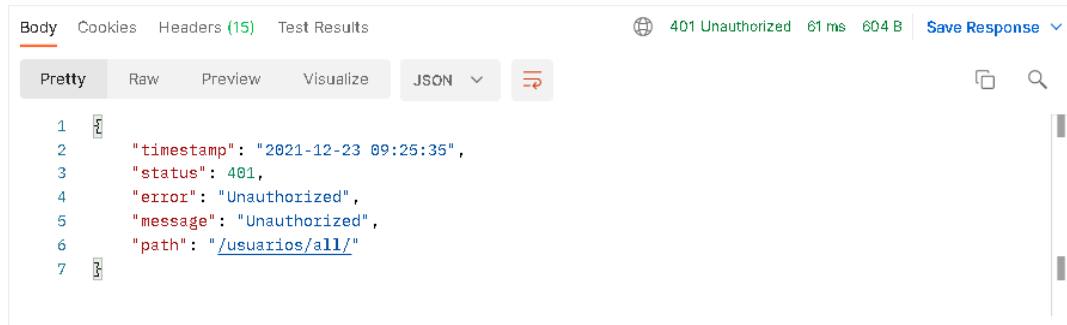
4. Observe que o Token foi gerado e enviado na resposta. Copie o Token da resposta porquê vamos precisar dele nas próximas requisições.
5. Caso o login falhe, você receberá o **status 401 (Unauthorized)**.

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

1. Na **Pasta Usuario**, clique nos 3 pontinhos ao lado do nome da pasta, para abrir o menu e clique na opção **Add Request**.
2. Configure a requisição conforme a imagem abaixo:



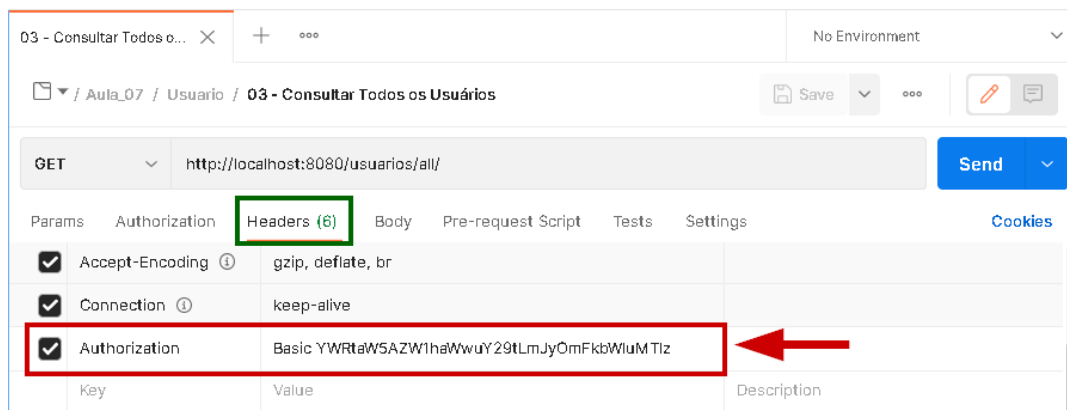
3. Como a segurança foi habilitada, caso você envie a requisição do jeito que está, você receberá o **status 401 (Unauthorized)**, como mostra a figura abaixo:




```
1 {
2   "timestamp": "2021-12-23 09:25:35",
3   "status": 401,
4   "error": "Unauthorized",
5   "message": "Unauthorized",
6   "path": "/usuarios/all/"
7 }
```

4. A explicação é simples: **Você precisa passar no cabeçalho da requisição o Token recebido no login.**

5. Clique na opção **Headers** e adicione a linha **Authorization**. Na coluna **Value** da linha insira o **Token** que você recebeu no Login, como mostra a figura abaixo:



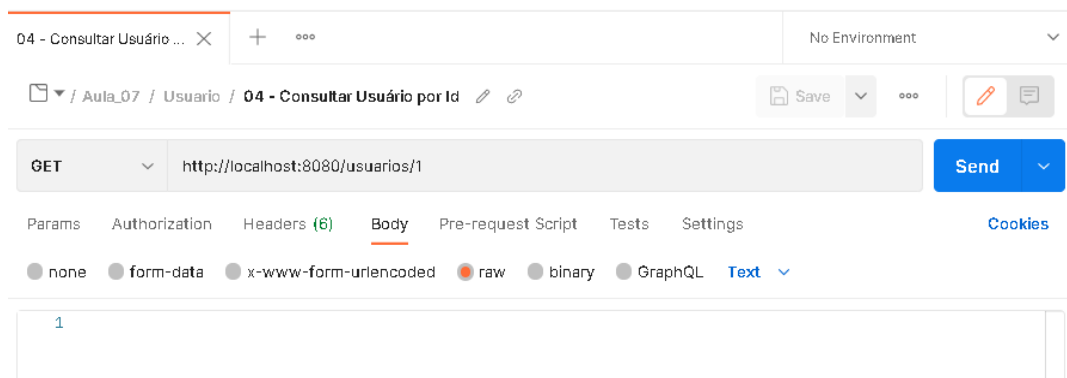
6. Agora a sua requisição funcionará!



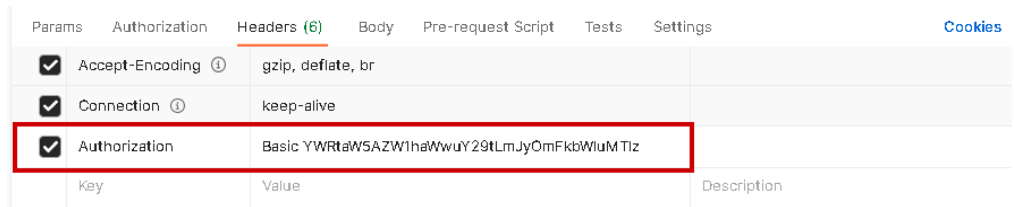
ALERTA DE BSM: Mantenha a Atenção aos Detalhes! Você deverá inserir o token no cabeçalho de todas requisições dos Recursos Postagem e Tema, caso contrário você receberá o status 401 (Unauthorized) em todas elas. Caso você efetue login com outro Usuário, o Token deve ser alterado.

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

1. Na **Pasta Usuario**, clique nos 3 pontinhos ao lado do nome da pasta, para abrir o menu e clique na opção **Add Request**.
2. Configure a requisição conforme a imagem abaixo:

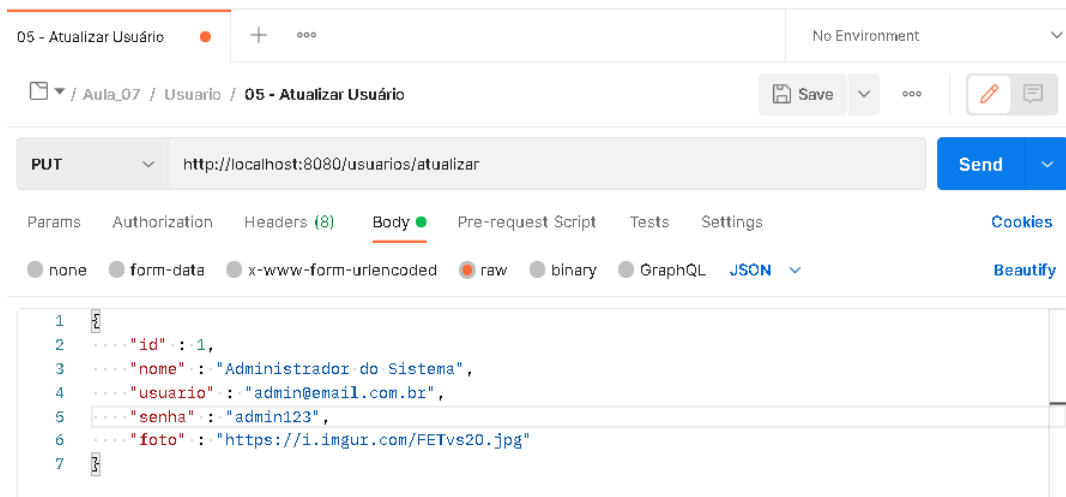


3. Clique na opção **Headers** e adicione a linha **Authorization**. Na coluna **Value** da linha insira o **Token** que você recebeu no Login, como mostra a figura abaixo:

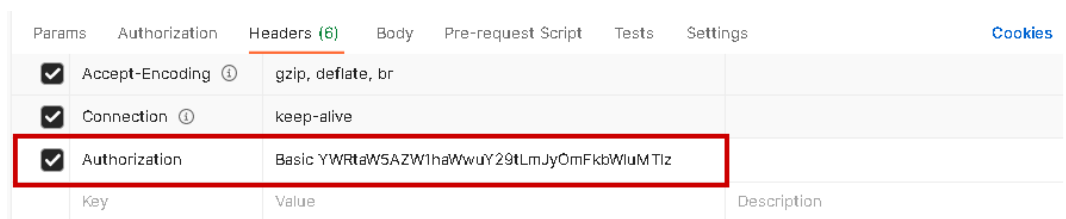


4.6. Criando a Request - Atualizar Usuário

1. Na **Pasta Usuário**, clique nos 3 pontinhos ao lado do nome da pasta, para abrir o menu e clique na opção **Add Request**.
2. Configure a requisição conforme a imagem abaixo:



3. 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.
4. Clique na opção **Headers** e adicione a linha **Authorization**. Na coluna **Value** da linha insira o **Token** que você recebeu no Login, como mostra a figura abaixo:



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

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:

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

No item marcado em vermelho na imagem acima, 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:

```
1 {
2   .... "id" : 1,
3   .... "titulo" : "Postagem 01 atualizado!",
4   .... "texto" : "Texto da Postagem 01 atualizado!",
5   .... "tema" : {
6     .... "id" : 2
7   },
8   .... "usuario" : {
9     .... "id" : 2
10  }
11 }
```

No item marcado em vermelho na imagem acima, 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.



DESAFIO: O que acontecerá se você inserir no JSON das requisições Cadastrar e Atualizar Postagem, no Objeto da Classe Usuario chamado usuario, um id que não existe? Insira no Atributo id, do Objeto Usuario, um id como 100, por exemplo, e veja o que acontece.



Passo 06 - Atualizar os Métodos post e put na Classe PostagemController

Se você fez o desafio acima, percebeu que estas implementações não conseguem checar se o **Objeto da Classe Usuario existe**, logo se você inserir um Objeto que não existe (um Id que não existe no Banco de dados), devido ao Relacionamento entre as Classes, será retornado o **HTTP Status 500 - Internal Server Error**.

Para evitar este erro, faremos alguns ajustes na **Classe PostagemController**.

6.1. Inserir uma Injeção de Dependência da Classe Usuario na Classe PostagemController

```
32
33 @Autowired
34 private PostagemRepository postagemRepository;
35
36 @Autowired
37 private TemaRepository temaRepository;
38
39 @Autowired
40 private UsuarioRepository usuarioRepository;
41
```

Linhas 39 e 40: Para termos acesso aos **Métodos das Classes Usuario e UsuarioController**, precisamos inserir uma Injeção de Dependência da Classe Usuario, logo abaixo da Injeção de Dependência da Classe Tema.

6.2. Atualização do Método post da Classe PostagemController

```
58
59 @PostMapping
60 public ResponseEntity<Postagem> post(@Valid @RequestBody Postagem postagem){
61     if (temaRepository.existsById(postagem.getTema().getId()) &&
62         usuarioRepository.existsById(postagem.getUsuario().getId()) )
63         return ResponseEntity.status(HttpStatus.CREATED)
64             .body(postagemRepository.save(postagem));
65
66     throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Tema ou Usuário não existem!", null);
67 }
68
```

Linhas 61 e 62: Através do Método **existsById(Long id)**, da Interface **UsuarioRepository** (Herança da Interface **JPA**), checamos se o id passado no Objeto usuario, da Classe **Usuario**, inserido no Objeto **postagem**, da Classe **Postagem**, existe, assim como fizemos na checagem da Classe **Tema**. Se ambos os Objetos existirem, a nova postagem será persistida no Banco de dados.

Para obter o id do usuario, utilizamos os Métodos **get** das 2 Classes:
postagem.getUsuario().getId()

Linhas 66: Através do Método **throw**, Caso Usuario e/ou Tema não existam, uma Exceção (Exception) é disparada com o **HTTP Status 400 (Bad Request)** e uma mensagem informando os possíveis motivos do erro.



[Documentação: existsById\(\)](#)

6.3. Atualização do Método put da Classe PostagemController

```
68
69 @PutMapping
70 public ResponseEntity<Postagem> put(@Valid @RequestBody Postagem postagem){
71     if (postagemRepository.existsById(postagem.getId())){
72
73         if (temaRepository.existsById(postagem.getTema().getId()) &&
74             usuarioRepository.existsById(postagem.getUsuario().getId()) )
75             return ResponseEntity.status(HttpStatus.OK)
76                 .body(postagemRepository.save(postagem));
77
78         throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Tema ou Usuário não existem!", null);
79     }
80 }
81
82 return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
83 }
84
85 }
```

Linhas 73 e 74: Através do Método **existsById(Long id)**, da Interface UsuarioRepository (Herança da Interface JPA), checamos se o id passado no Objeto usuario, da Classe Usuario, inserido no Objeto postagem, da Classe Postagem, existe, assim como fizemos na checagem da Classe Tema. Se ambos os Objetos existirem, a nova postagem será persistida no Banco de dados.

Para obter o id do usuario, utilizamos os Métodos **get** das 2 Classes:
postagem.getUsuario().getId().

Linhas 78: Através do Método **throw**, Caso Usuario e/ou Tema não existam, uma Exceção (Exception) é disparada com o **HTTP Status 400 (Bad Request)** e uma mensagem informando os possíveis motivos do erro.



[Código fonte do projeto](#)

Segurança do Blog Pessoal finalizada!!!