

Desenvolvimento de APIs REST

11 - E-mails e Upload de Arquivos

- Envio de E-mails
- Controller - Gerando URIs
- Upload de Arquivos
- Acesso API's Externas

Criar recursos para envio de e-mails



Em nossas aplicações provavelmente vamos precisar da funcionalidade de envio de e-mails, vamos criar um exemplo utilizando o Spring Boot. **Vamos utilizar o projeto da aula anterior sobre service e dto.**

O primeiro passo é adicionar a dependência abaixo no pom.xml

Podemos fazer isso copiando do site [www.mvnrepository.com](https://mvnrepository.com) ou clicando com o botão direito sobre o projeto **Spring – Add Starters** conforme imagens ao lado.

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-mail</artifactId>
</dependency>
```

Home » org.springframework.boot » spring-boot-starter-mail » 3.2.1



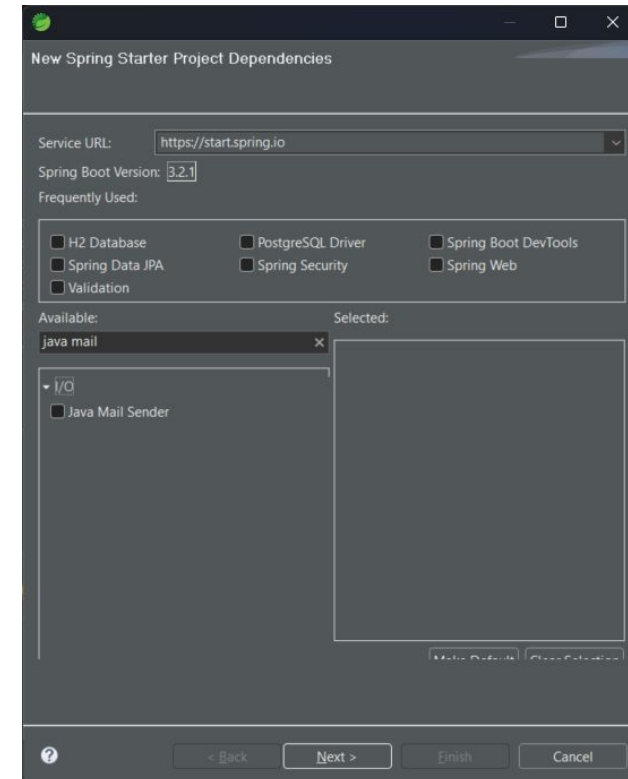
Spring Boot Starter Mail » 3.2.1

Starter for using Java Mail and Spring Framework's email sending support

License	Apache 2.0
Tags	spring framework mail starter
Organization	VMware, Inc.
HomePage	https://spring.io/projects/spring-boot
Date	Dec 21, 2023
Files	pom (2 KB) jar (4 KB) View All
Repositories	Central
Ranking	#489 in MvnRepository (See Top Artifacts)
Used By	965 artifacts

Maven Gradle Gradle (Short) Gradle (Kotlin) SBT Ivy Grape Leiningen Buildr

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-mail -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-mail</artifactId>
  <version>3.2.1</version>
</dependency>
```



Criar recursos para envio de e-mails



Java Mail

É uma biblioteca que permite de forma fácil realizar configurações para envio de emails.

Criar a classe **MailConfig** que será responsável pelas configurações de envio. Anotar a classe com a anotação **@Configuration**

```
@Configuration
public class MailConfig {
    @Autowired
    private JavaMailSender javaMailSender;

    public void sendEmail(String para, String assunto, String texto) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom("bulinha@gmail.com");
        message.setTo(para);
        message.setSubject(assunto);
        message.setText("Dados da inscrição: \n" + texto + "\n\nSerratec Residência de Software");
        javaMailSender.send(message);
    }
}
```

Configuração de e-mail



Vamos inserir as propriedades de conexão ao servidor SMTP utilizando o Gmail

```
spring.datasource.url=jdbc:postgresql://localhost:5432/projeto
spring.datasource.username=postgres
spring.datasource.password=postgres
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=none
spring.jackson.serialization.fail-on-unknown-properties=false
```

```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=seuemail@gmail.com
spring.mail.password=suasenha
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.starttls.required=true
```

```
auth.jwt-secret=EAssimQueMeuFuscaAnda_EAssimQueEleVaiParar
auth.jwt-expiration-miliseg=120000
```

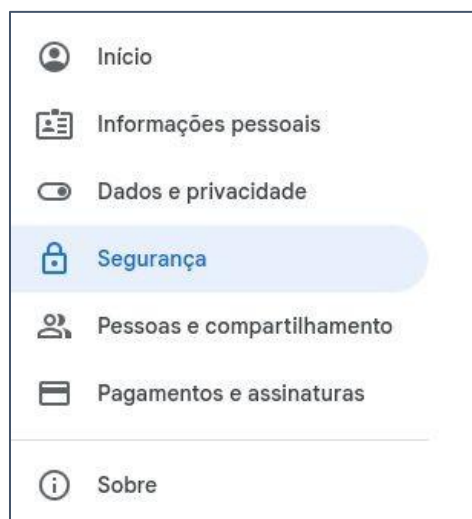
- smtp - Simple Mail Transfer Protocol: protocolo de transferência de correio eletrônico simples
- tls - Transport Layer Security: protocolo de segurança proprietário projetado para fornecer segurança nas comunicações

Utilizando o GMail para envio de Emails

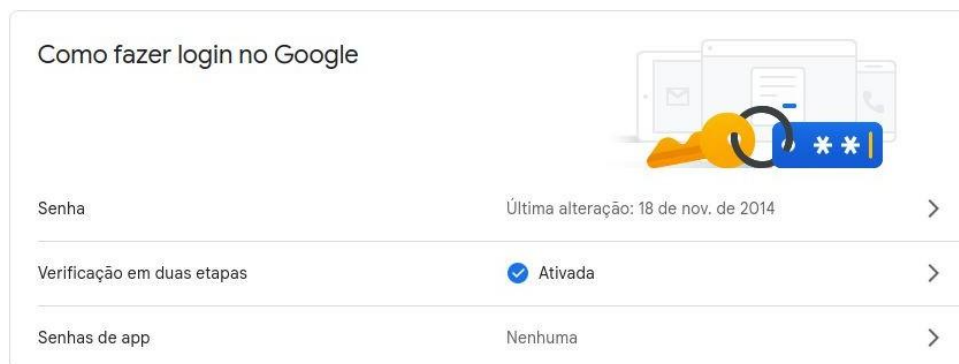


1) Acesse conta do Google <https://myaccount.google.com/>

No menu à esquerda, acesse Segurança



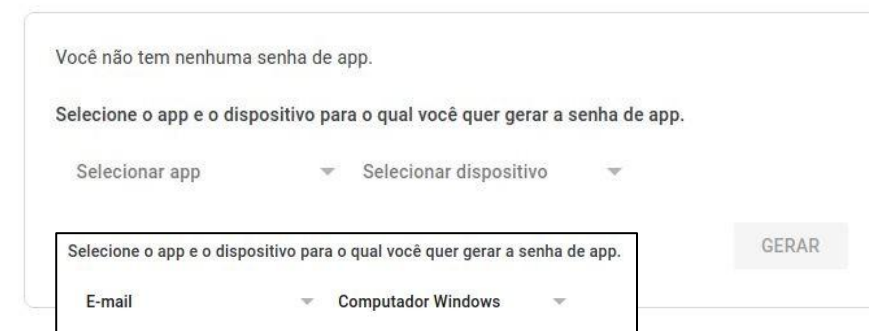
2) Encontre a seção “Como fazer login no Google” e acesse Senhas de App



3) Selecione em app “E-mail” e em dispositivo “Computador Windows”. Clique em GERAR.

← Senhas de app

Senhas de app permitem que você faça login na sua Conta do Google a partir de apps em dispositivos que não sejam compatíveis com a verificação em duas etapas. Como só será necessário informar a senha uma vez, você não precisa memorizá-la. [Saiba mais](#)



Utilizando o GMail para envio de Emails



Copie a senha gerada (texto em amarelo) antes de clicar em CONCLUÍDO, pois após isso não será mais possível copiá-la, apenas apagando e gerando uma nova.

Senha de app gerada

Sua senha de app para computador Windows

ewzl mxkl qzbo kxsl

Como usar

1. Abra o app "Mail".
2. Abra o menu "Configurações".
3. Selecione "Contas" e selecione sua Conta do Google.
4. Substitua sua senha pela senha de 16 caracteres mostrada acima.

Assim como sua senha normal, esta senha de app concede acesso total à sua Conta do Google. Não é necessário memorizá-la, por isso não a anote ou a compartilhe com outras pessoas.

[Saiba mais](#)

CONCLUÍDO

Add your Google account

Enter the information below to connect to your Google account.

Email address

Password

☐ Include your Google contacts and calendars

Suas senhas de app

Nome	Criada	Usada pela última vez em
E-mail no meu Computador Windows	20:30	-

Selecione o app e o dispositivo para o qual você quer gerar a senha de app.

[Selecionar app](#) [Selecionar dispositivo](#)

GERAR

Use esta senha no arquivo application.properties na propriedade:
spring.mail.password

Envio de Email



Inseri a linha “mailConfig...” na classe **UsuarioService** como no código abaixo
Não esquecer de incluir a dependência para MailConfig

```
public UsuarioDTO inserir(UsuarioInserirDTO user) throws EmailException {
    if (!user.getSenha().equalsIgnoreCase(user.getConfirmaSenha())) {
        throw new SenhaException("Senha e Confirma Senha não são iguais");
    }
    if (usuarioRepository.findByEmail(user.getEmail()) != null) {
        throw new EmailException("Email já existente");
    }
    Usuario usuario = new Usuario();
    usuario.setNome(user.getNome());
    usuario.setEmail(user.getEmail());
    usuario.setSenha(passwordEncoder.encode(user.getSenha()));

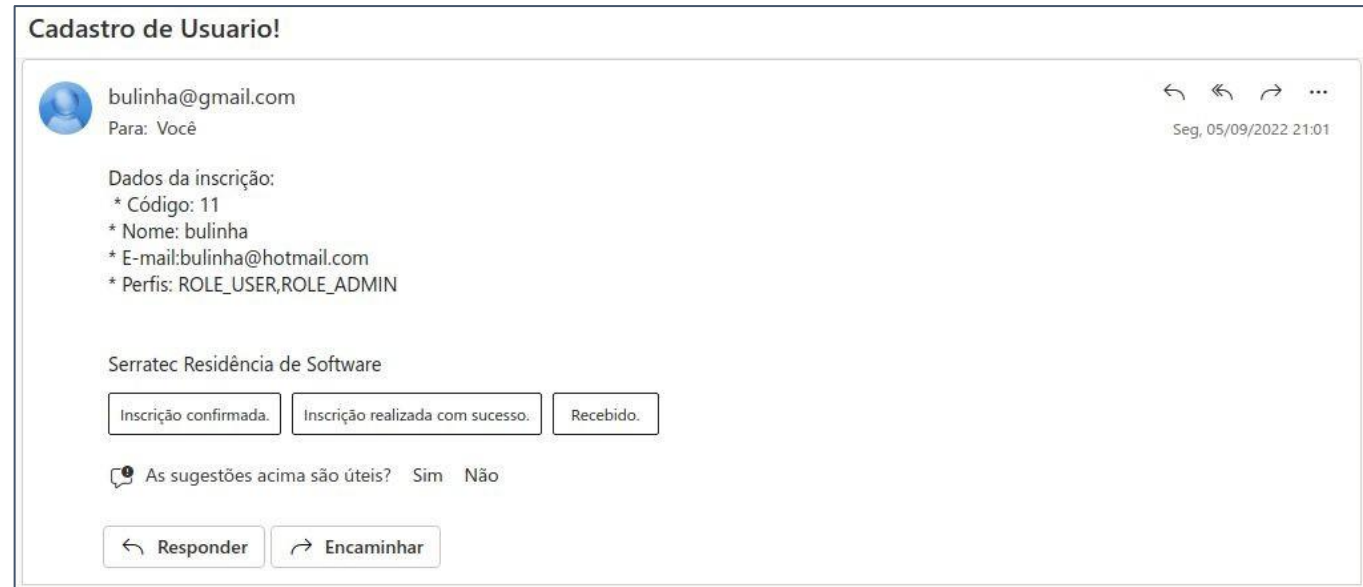
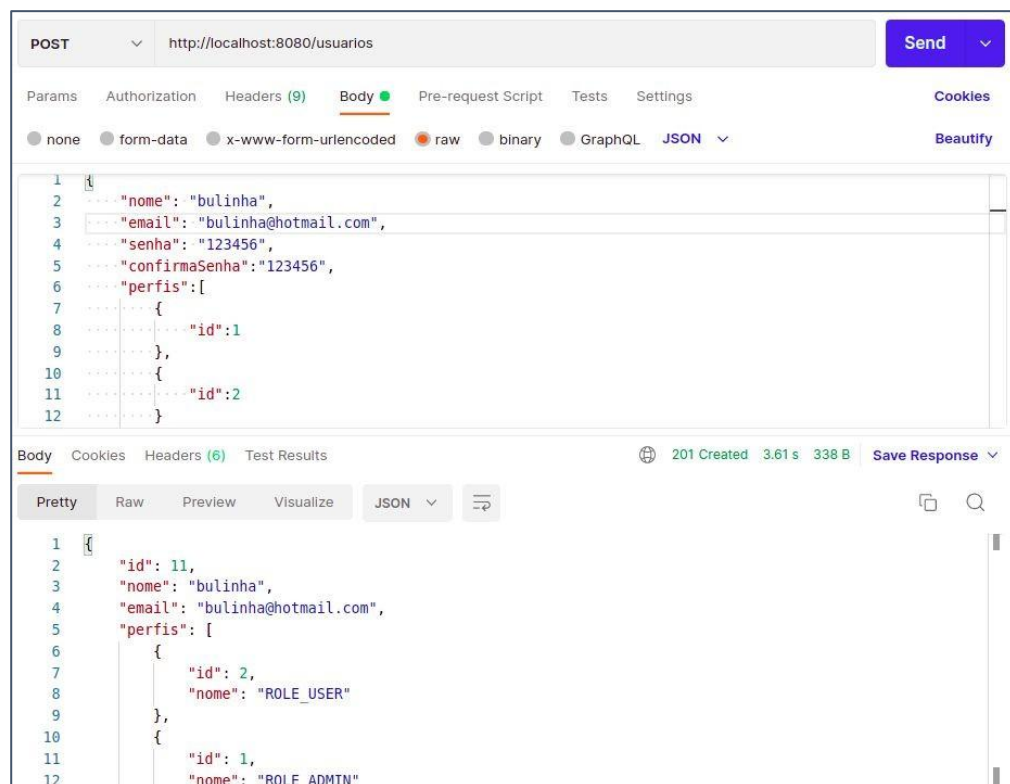
    Set<UsuarioPerfil> perfis = new HashSet<>();
    for(Perfil perfil: user.getPerfis()) {
        perfil = perfilService.buscar(perfil.getId());
        UsuarioPerfil usuarioPerfil = new UsuarioPerfil(usuario, perfil, LocalDate.now());
        perfis.add(usuarioPerfil);
    }
    usuario.setUsuarioPerfis(perfis);
    usuario = usuarioRepository.save(usuario);

    mailConfig.sendEmail(usuario.getEmail(), "Cadastro de Usuario!", usuario.toString());
    return new UsuarioDTO(usuario);
}
```

```
@Override
public String toString() {
    return "* Código: " + id +
        "\n* Nome: " + nome +
        "\n* E-mail: " + email +
        "\n* Perfis: " +
            usuarioPerfis
                .stream()
                .map(up -> up.getId().getPerfil().getNome())
                .collect(Collectors.joining(", "));
}
```

Vamos adicionar o método **toString** na classe **Usuario** para retorno dos dados do Usuario

Testar com o Postman



Criar mais um arquivo de migração: V05cria_tabela_foto.sql

Upload de Arquivos

```
create table foto (  
  id_foto serial primary key,  
  dados bytea,  
  tipo varchar(100),  
  nome varchar(100),  
  id_funcionario bigint,  
  foreign key (id_funcionario) references funcionario(id_funcionario)  
);
```

Criar Classes de Domínio



Criar a entidade foto, não esquecendo de criar os gets e sets

```
@Entity
public class Foto {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_foto")
    private Long id;

    @Lob
    @Type(type="org.hibernate.type.BinaryType")
    private byte[] dados;

    private String tipo;
    private String nome;

    @OneToOne
    @JoinColumn(name="id_funcionario")
    private Funcionario funcionario;

    public Foto() {
    }

    public Foto(Long id, byte[] dados, String tipo, String nome, Funcionario funcionario) {
        this.id = id;
        this.dados = dados;
        this.tipo = tipo;
        this.nome = nome;
        this.funcionario = funcionario;
    }
}
```

Repositorys e DTO



DTO de funcionário com o atributo para a URL da foto

```
public class FuncionarioDTO {
    private String nome;
    private Double salario;
    private String url;

    @JsonFormat(pattern = "dd/MM/yyyy")
    private LocalDate dataNascimento;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public Double getSalario() {
        return salario;
    }

    public void setSalario(Double salario) {
        this.salario = salario;
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public LocalDate getDataNascimento() {
        return dataNascimento;
    }

    public void setDataNascimento(LocalDate dataNascimento) {
        this.dataNascimento = dataNascimento;
    }
}
```

Repositório de Foto com método de busca de foto por funcionário

```
@Repository
public interface FotoRepository extends JpaRepository<Foto, Long> {
    public Optional<Foto> findByFuncionario(Funcionario funcionario);
}
```

A anotação de JsonFormat, serve para realizar a conversão da data para string (json) e vice versa.

```
@JsonFormat(pattern = "dd/MM/yyyy")
```

Ela também deve ser colocada no atributo dataNascimento na entidade Funcionario, para quando for fazermos a inclusão.

Service de Foto



```
@Service
public class FotoService {
    @Autowired
    private FotoRepository fotoRepository;

    public Foto inserir(Funcionario funcionario, MultipartFile file) throws IOException{
        Foto foto = new Foto();
        foto.setNome(file.getName());
        foto.setTipo(file.getContentType());
        foto.setDados(file.getBytes());
        foto.setFuncionario(funcionario);
        return fotoRepository.save(foto);
    }

    @Transactional
    public Foto buscarPorIdFuncionario(Long id) {
        Funcionario funcionario = new Funcionario();
        funcionario.setId(id);
        Optional<Foto> foto = fotoRepository.findByFuncionario(funcionario);
        if (!foto.isPresent()){
            return null;
        }
        return foto.get();
    }
}
```

Service de Foto que insere a foto relacionada ao funcionário a partir do objeto MultipartFile.

MultipartFile é uma classe do Spring utilizada quando uma requisição é feita utilizando Multipart/Form-Data.

<https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Metods/POST>

Método para buscar uma foto a partir de um ID de funcionário.

A anotação @Transactional é utilizada para encapsular transações no banco de dados. Deve ser utilizada quando há a necessidade de atualizar diversas tabelas (e em caso de erro, desfazer as atualizações) ou trabalhar com arquivo grandes armazenados no banco de dados (no caso de fotos). Mais informações em <https://www.devmedia.com.br/conheca-o-spring-transactional-annotations/32472>

Service de Funcionário



```
@Service
public class FuncionarioService {
    @Autowired
    private FuncionarioRepository funcionarioRepository;
    @Autowired
    private FotoService fotoService;

    public List<FuncionarioDTO> listar() {
        List<FuncionarioDTO> funcionarioDTOS = funcionarioRepository
            .findAll()
            .stream()
            .map( f-> adicionarImagemUri(f) ).collect(Collectors.toList());
        return funcionarioDTOS;
    }

    public FuncionarioDTO adicionarImagemUri(Funcionario funcionario) {
        URI uri = ServletUriComponentsBuilder
            .fromCurrentContextPath()
            .path("/{funcionarios}/{id}/foto")
            .buildAndExpand(funcionario.getId())
            .toUri();

        FuncionarioDTO dto = new FuncionarioDTO();
        dto.setNome(funcionario.getNome());
        dto.setDataNascimento(funcionario.getDataNascimento());
        dto.setSalario(funcionario.getSalario());
        dto.setUrl(uri.toString());
        return dto;
    }
}
```

Método **adicionarImagemUri** cria um **FuncionarioDTO** e utiliza o componente **ServletUriComponentsBuilder** para criar a url para o método do controller que retornará a foto do funcionário.

Os métodos **listar**, **buscar** e **inserir**, utilizam o **adicionarImagemUri** para criar o objeto **FuncionarioDTO** com a url da foto.

```
public FuncionarioDTO buscar(Long id) {
    Optional<Funcionario> funcionario = funcionarioRepository.findById(id);
    return adicionarImagemUri(funcionario.get());
}

public FuncionarioDTO inserir(Funcionario funcionario, MultipartFile file) throws IOException {
    funcionario = funcionarioRepository.save(funcionario);
    fotoService.inserir(funcionario, file);
    return adicionarImagemUri(funcionario);
}
}
```

Controller



```
@RequestMapping("/funcionarios")
public class FuncionarioController {
    @Autowired
    private FuncionarioRepository funcionarioRepository;
    @Autowired
    private FuncionarioService funcionarioService;
    @Autowired
    private FotoService fotoService;

    @GetMapping
    public List<FuncionarioDTO> listar(){
        return funcionarioService.listar();
    }

    @GetMapping("/{id}/foto")
    public ResponseEntity<byte[]> buscarFoto(@PathVariable Long id) {
        Foto foto = fotoService.buscarPorIdFuncionario(id);
        HttpHeaders headers = new HttpHeaders();
        headers.add("Content-type", foto.getTipo());
        headers.add("Content-length", String.valueOf(foto.getDados().length));
        return new ResponseEntity<>(foto.getDados(), headers, HttpStatus.OK);
    }

    @GetMapping("/{id}")
    public FuncionarioDTO buscar(@PathVariable Long id) {
        return funcionarioService.buscar(id);
    }

    @PostMapping(consumes = {MediaType.MULTIPART_FORM_DATA_VALUE})
    public FuncionarioDTO inserir(@RequestPart MultipartFile file, @RequestPart Funcionario funcionario) throws IOException{
        return funcionarioService.inserir(funcionario, file);
    }
}
```

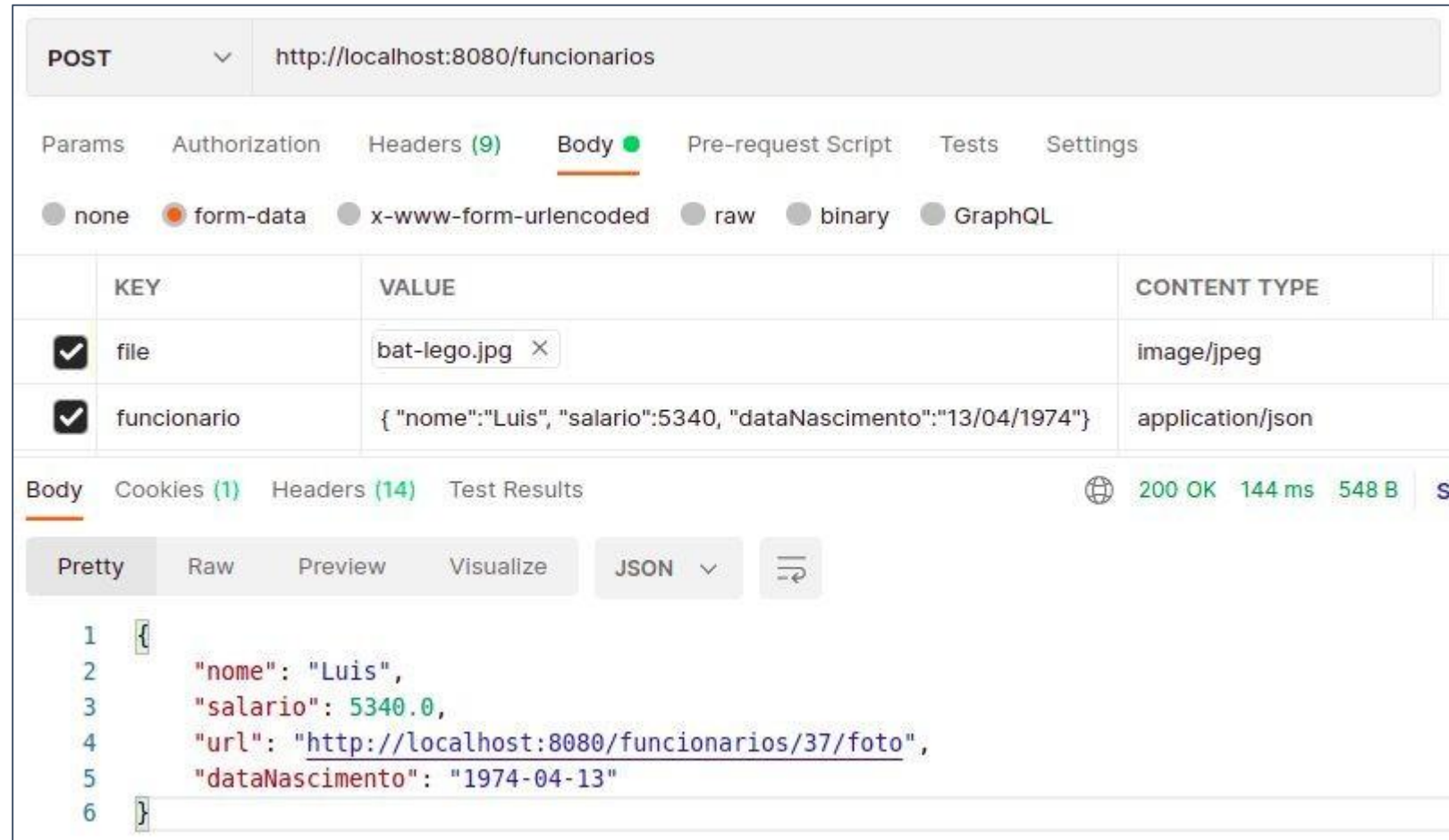
O método **buscarFoto** utiliza o objeto **HttpHeaders** para incluir na resposta da requisição, as informações de cabeçalho: Content-type e Content-length, contendo respectivamente o tipo de conteúdo (ex: image/jpeg) e o tamanho da resposta em bytes.

Como a requisição irá enviar tanto o json referente ao funcionário quanto a imagem da foto, é necessário que esta requisição seja do tipo Multipart/Form-Data. A anotação **PostMapping** deve indicar que ela irá “consumir” esse tipo de dados

Realizando Post no Postman



1. No final dos nomes das colunas há o símbolo "...", clique nele para selecionar as colunas a serem exibidas, selecione a coluna content type
2. Informar que é do tipo form-data
3. inserir o atributo file na coluna key (ao passar o mouse sobre o campo há uma lista de opções para selecionar o tipo de parâmetro, selecione a opção **file**)
4. no campo value irá aparecer um botão "select file", clique e selecione o arquivo que deseja
5. para o atributo funcionário, preenche a coluna value com o json a ser enviado
6. preencha os valores da coluna content type com image/jpeg (verifique o tipo da imagem que você selecionou) e application/json
7. Ao enviar ele retornará o json contendo o nome do funcionário inserido e a url da imagem



Realizando Get no Postman



The screenshot shows the Postman interface with a GET request configured. The URL is `http://localhost:8080/funcionarios/37/foto`. The 'Params' tab is active, showing a table for Query Params.

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Below the table, the 'Body' tab is active, showing a status bar with a globe icon, `403 Forbidden`, `461 ms`, `401 B`, and a `Save Response` dropdown. The response body is displayed in 'Text' format, showing the number `1`.

É preciso alterar a configuração de segurança para “liberar” este novo caminho.

Como parte do caminho é dinâmico, é necessário utilizar expressões regulares.

Realizando Get no Postman



Para incluir o path da foto, devemos usar a expressão regex `{\\d+}`, que indica uma string contendo apenas dígitos. Assim o “pattern” para identificar o endereço `/funcionarios/37/foto` seria `/funcionarios/{\\d+}/foto`.

```
http.authorizeHttpRequests()
    .requestMatchers("/public/**").permitAll()
    .requestMatchers("/funcionarios").permitAll()
    .requestMatchers("/funcionarios/{\\d+}/foto").permitAll()
    .requestMatchers("/funcionarios/salarios-por-idade").permitAll()
    .requestMatchers(HttpMethod.GET, "/funcionarios/salario", "/funcionarios/pagina", "/funcionarios/nome").hasAuthority("ADMIN")
    .requestMatchers(HttpMethod.GET, "/usuarios").hasAnyAuthority("ADMIN", "USER")
    .requestMatchers(HttpMethod.POST, "/usuarios").hasAuthority("ADMIN")
    .anyRequest().authenticated()
```

Mais informações sobre antMatchers do Spring Security

<https://bushansirgur.in/everything-need-to-know-about-matchers-methods-in-spring-security/> Mais informações sobre expressões regulares

<http://turing.com.br/material/regex/introducao.html>

Realizando Get no Postman



Após tudo configurado, se acessarmos a url da imagem no postman veremos a imagem retornada no body e no headers o content-type e content-length


GET `http://localhost:8080/funcionarios/37/foto` **Send**

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies (1) Headers (14) Test Results **200 OK 6.39 s 25.18 KB Save Response**



Body Cookies (1) Headers (14) Test Results **200 OK 6.39 s 25.18 KB Save Response**

Header	Value
X-Content-Type-Options	nosniff
X-XSS-Protection	1; mode=block
Cache-Control	no-cache, no-store, max-age=0, must-revalidate
Pragma	no-cache
Expires	0
X-Frame-Options	DENY
Content-Type	image/jpeg
Content-Length	25361
Date	Thu, 08 Sep 2022 03:31:07 GMT
Keep-Alive	timeout=60
Connection	keep-alive

Acessando API's



Webservice gratuito para consultar CEPs

JSON

URL: viacep.com.br/ws/01001000/json/

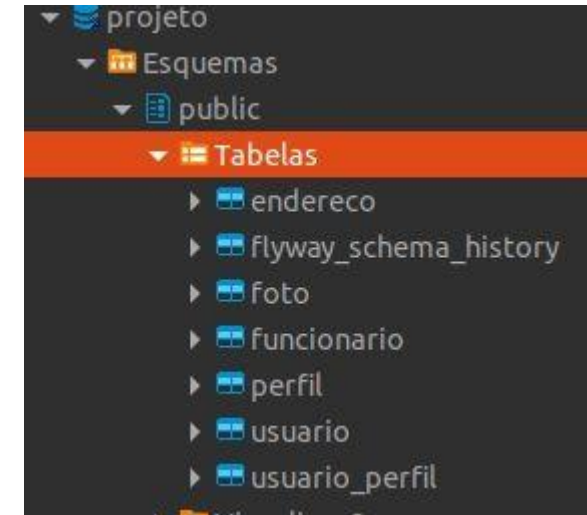
```
{
  "cep": "01001-000",
  "logradouro": "Praça da Sé",
  "complemento": "lado ímpar",
  "bairro": "Sé",
  "localidade": "São Paulo",
  "uf": "SP",
  "ibge": "3550308",
  "gia": "1004",
  "ddd": "11",
  "siafi": "7107"
}
```

Preparando a aplicação



Vamos inserir o script do banco de dados para criar uma tabela para armazenar os endereços dos usuários

```
CREATE TABLE endereco (  
  id_endereco serial PRIMARY KEY,  
  cep varchar(10),  
  logradouro varchar(50),  
  complemento varchar(30),  
  bairro varchar(40),  
  localidade varchar(40),  
  uf varchar(2),  
  ibge integer  
);  
  
ALTER TABLE usuario  
ADD COLUMN id_endereco bigint,  
ADD CONSTRAINT fk_id_endereco FOREIGN KEY (id_endereco)  
REFERENCES endereco;
```



Criar Entidade e Repositorio para Endereco



Criar a entidade **Endereco**, tomaremos por base para a seleção de atributos, os mesmos retornados pela api do **viacep** (vamos usar apenas os atributos essenciais):

```
@Entity
public class Endereco {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_endereco")
    private Long id;

    private String cep; private
    String logradouro; private
    String complemento; private
    String bairro; private
    String localidade; private
    String uf;
    private Long ibge;

    //gets e sets
}
```

Criar o repósitorio **EnderecoRepository** com um método para pesquisa de endereço pelo CEP

```
@Repository
public interface EnderecoRepository extends JpaRepository<Endereco, Long>{

    public Endereco findByCep(String cep) ;

}
```

JSON

URL: viacep.com.br/ws/01001000/json/

```
{
  "cep": "01001-000",
  "logradouro": "Praça da Sé",
  "complemento": "lado ímpar",
  "bairro": "Sé",
  "localidade": "São Paulo",
  "uf": "SP",
  "ibge": "3550308",
  "gia": "1004",
  "ddd": "11",
  "siafi": "7107"
}
```

Service com RestTemplate



```
@Service
public class EnderecoService {
    @Autowired
    private EnderecoRepository enderecoRepository;
    public EnderecoDTO buscar(String cep) {
        Optional<Endereco> endereco =
            Optional.ofNullable(enderecoRepository.findByCep(cep)); if (endereco.isPresent()) {
            return new EnderecoDTO(endereco.get());
        } else {
            RestTemplate restTemplate = new RestTemplate();
            String uri = "http://viacep.com.br/ws/"+cep+"/json";
            Optional<Endereco> enderecoViaCep = Optional.ofNullable(restTemplate.getForObject(uri, Endereco.class));
            if (enderecoViaCep.get().getCep() != null) {
                String cepSemTrace = enderecoViaCep.get().getCep().replaceAll("-", "");
                enderecoViaCep.get().setCep(cepSemTrace);
            }
            return inserir(enderecoViaCep.get());
        } else {
            return null;
        }
    }

    private EnderecoDTO inserir(Endereco endereco) {
        return new EnderecoDTO(enderecoRepository.save(endereco));
    }
}
```

```
public class EnderecoDTO {

    private String cep;
    private String logradouro;
    private String complemento;
    private String bairro;
    private String localidade;
    private String uf;

    public EnderecoDTO(Endereco endereco) {
        this.cep=endereco.getCep();
        this.logradouro=endereco.getLocalidade();
        this.complemento=endereco.getComplemento();
        this.bairro=endereco.getBairro();
        this.localidade=endereco.getLocalidade();
        this.uf=endereco.getUf();
    }

    public String getCep() {
        return cep;
    }

    //gets e sets
}
```

RestTemplate é a classe do Spring responsável por acessar serviços REST externos a nossa aplicação. Mais informações em <https://www.baeldung.com/rest-template>

Criar o Controller



Criar o controler EnderecoController. Verificar se é necessário liberar o acesso para o path na configuração de segurança.

```
@RestController
@RequestMapping("/enderecos")
public class EnderecoController {
    @Autowired
    private EnderecoService enderecoService;

    @GetMapping("{cep}")
    public ResponseEntity<EnderecoDTO> buscar(@PathVariable String cep) {
        EnderecoDTO enderecoDTO = enderecoService.buscar(cep);
        if (enderecoDTO==null) {
            return ResponseEntity.notFound().build();
        } else {
            return ResponseEntity.ok().body(enderecoDTO);
        }
    }
}
```

```
.antMatchers("/enderecos/**").permitAll()
```

Teste no Postman



The screenshot shows the Postman interface for a GET request to `http://localhost:8080/enderecos/25710170`. The request is successful, returning a 200 OK status with a response time of 398 ms and a body size of 555 B. The response is displayed in the 'Body' tab, formatted as JSON.

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body

```
1 {  
2   "cep": "25710170",  
3   "logradouro": "Petrópolis",  
4   "complemento": "",  
5   "bairro": "Itamarati",  
6   "localidade": "Petrópolis",  
7   "uf": "RJ"  
8 }
```