Deploy de back-end em Spring utilizando Heroku

O que é o Heroku?

O Heroku é uma plataforma de cloud que oferece “Platform as a Service”, ou seja, ele permite que você hospede suas aplicações em um ambiente facilmente escalável e com suporte a várias tecnologias. Ele tem um plano free, que é indicado para testes, e opções pagas com mais funcionalidades e suporte.

O que veremos por aqui?

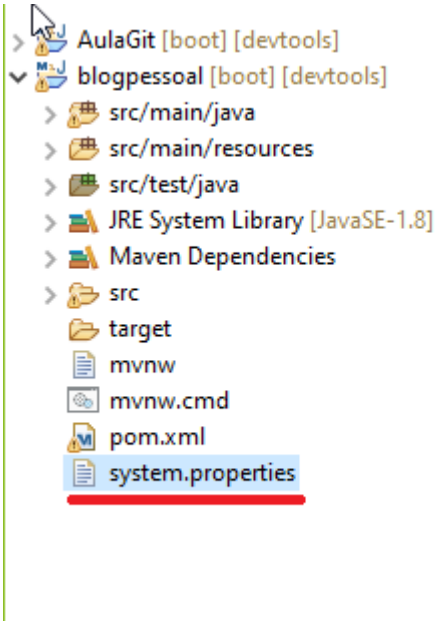
Esse documento é um passo a passo para você subir (deploy) o sua API criada em SPRING gratuitamente para o Heroku, que é uma aplicação de hospedagem de site na web, isso irá gerar um link de acesso a sua página que poderá ser acessadoem qualquer lugar. Para realizar esse deploy vamos precisar fazer algumas modificações em nosso projeto e principalmente já **criar uma conta no Heroku** através desse endereço: https://www.heroku.com.

#01 Passo

Crie a conta no heroku

#02 Passo

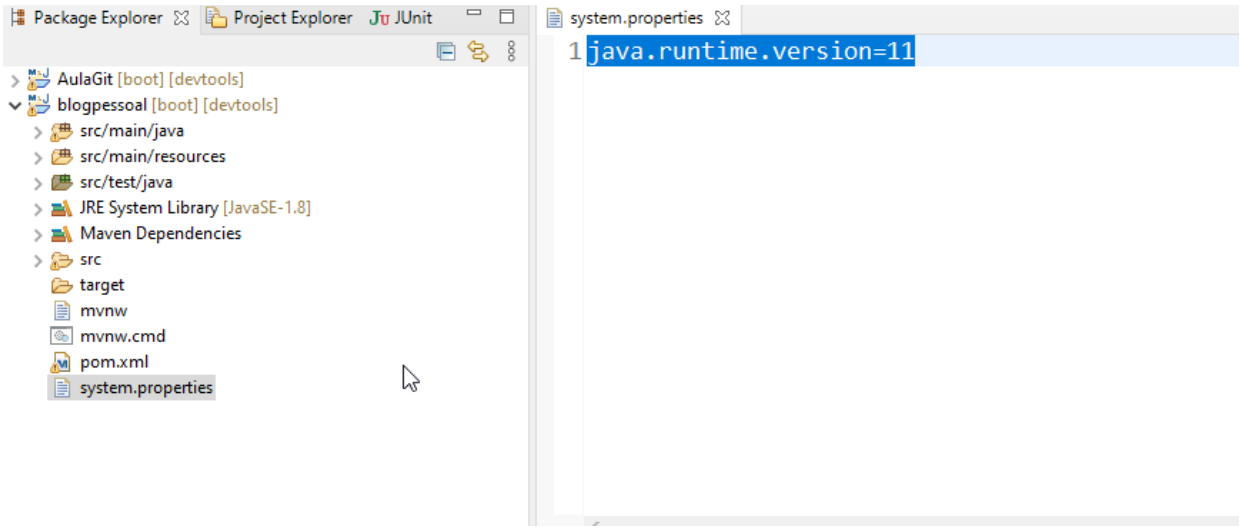
Crie um arquivo `system.properties` na raiz do projeto.



coloque a seguinte informação:

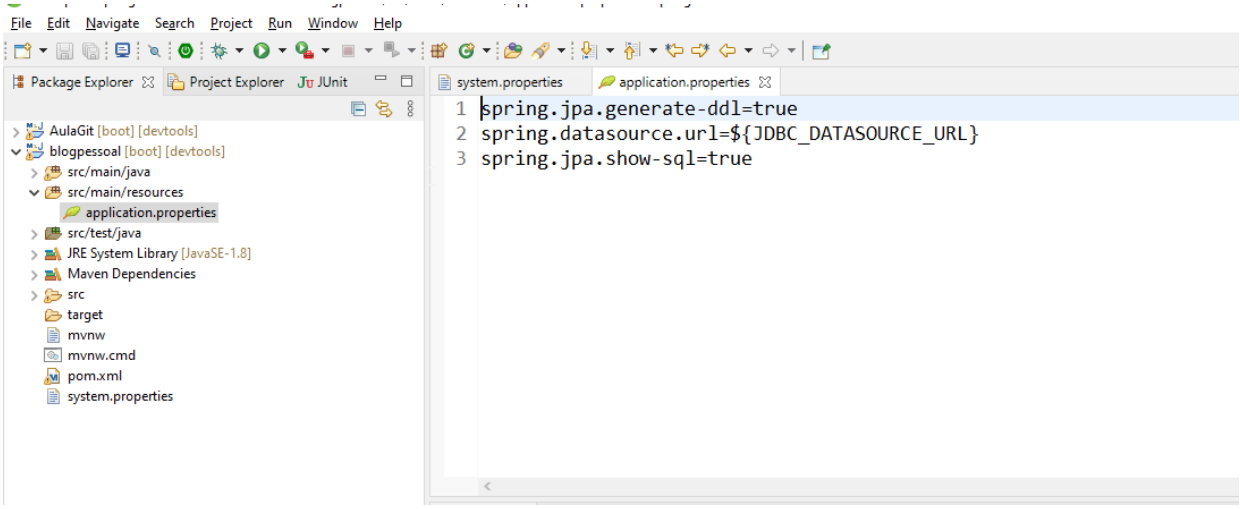
```
java.runtime.version=11
```

indique a versão do java do seu projeto.



#03 Passo

Substitua o conteúdo do arquivo application.properties, para:




```
spring.jpa.generate-ddl=true
spring.datasource.url=${JDBC_DATASOURCE_URL}
spring.jpa.show-sql=true
```

#04 Passo

Em **BasicSecurityConfig** vamos colocar o seguinte trecho de código para definir um usuario padrão em memoria.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.authentication.builde
import org.springframework.security.config.annotation.web.builders.HttpSecu
import org.springframework.security.config.annotation.web.configuration.Ena
import org.springframework.security.config.annotation.web.configuration.Web
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@EnableWebSecurity
public class BasicSecurityConfig extends WebSecurityConfigurerAdapter {

    private @Autowired UserDetailsServiceImpl service;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exce
        auth.inMemoryAuthentication()
        .withUser("admin").password(passwordEncoder().encode("admin")).auth

        auth.userDetailsService(service);
```

```
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
        .antMatchers("/usuarios/cadastrar").permitAll()
        .antMatchers("/usuarios/logar").permitAll()
        .anyRequest().authenticated()
        .and().httpBasic()
        .and().sessionManagement().sessionCreationPolicy(SessionCreationPol
        .and().cors()
        .and().csrf().disable();
    }


}
```

```
11 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
12 import org.springframework.security.crypto.password.PasswordEncoder;
13
14 @EnableWebSecurity
15 public class BasicSecurityConfig extends WebSecurityConfigurerAdapter {
16
17     private @Autowired UserDetailsServiceImpl service;
18
19     @Override
20     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
21         auth.inMemoryAuthentication()
22         .withUser("admin").password(passwordEncoder().encode("admin")).authorities("ROLE_ADMIN");
23
24         auth.userDetailsService(service);
25     }
26
```

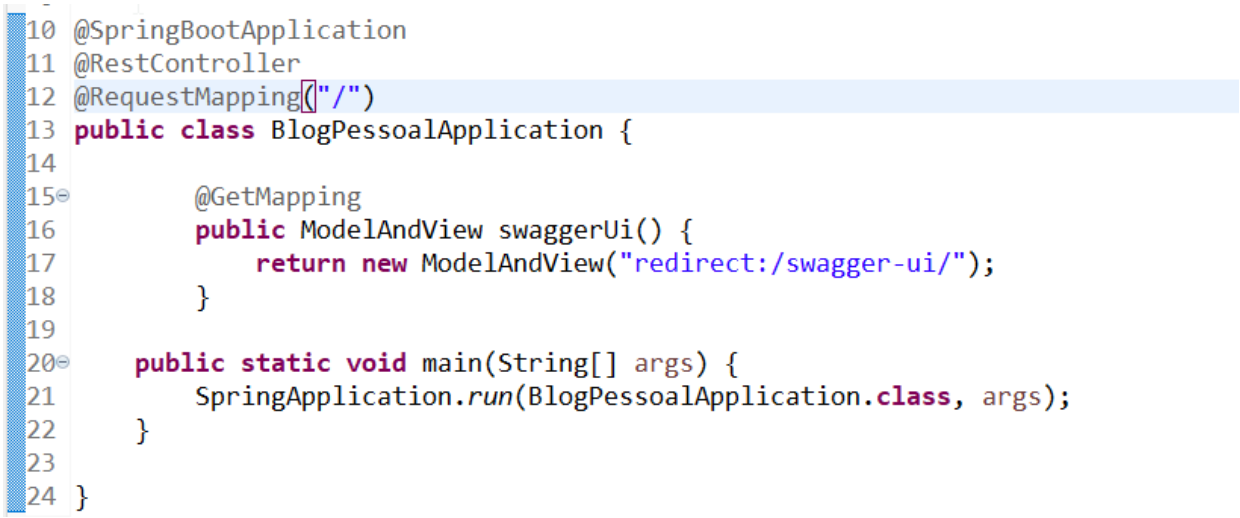
#05 Passo

Em **Main** vamos colocar o seguinte trecho de código para uma rota padrão para o swagger localhost:8080.

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

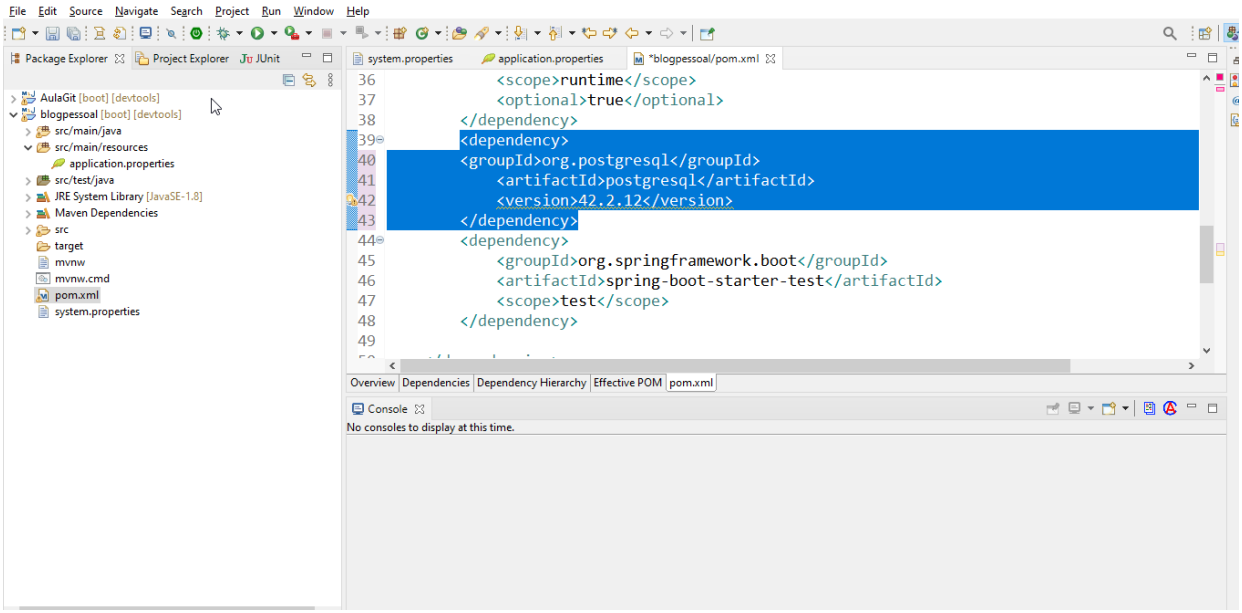
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.servlet.ModelAndView;

```
@SpringBootApplication
@RestController
@RequestMapping("/")
public class BlogPessoalApplication {

        @GetMapping
        public ModelAndView swaggerUi() {
            return new ModelAndView("redirect:/swagger-ui/");
        }

    public static void main(String[] args) {
        SpringApplication.run(BlogPessoalApplication.class, args);
    }

}
```



#06 Passo

Abra o **pom.xml** e substitua a dependência do MySql por essa dependência:




```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.2.12</version>
</dependency>
```

#07 Passo

Pronto, agora só precisamos abrir a **pasta que contém o arquivo pom.xml** no terminal de sua escolha e digite os seguintes comandos para criar um repositório no git:

git init
git add .
git commit -m “mensagem”

#09 Passo

Agora precisamos configurar o Heroku no terminal, porém antes de iniciar precisamos instalar o heroku através do pacote npm:

```
npm i -g heroku
```

Agora é só fazer o login no heroku e continuar as configurações:

```
heroku login
```

Após a execução desse comando será aberta em seu navegador uma página da Heroku com um botão para você logar, clique nele, volte para o terminal e prossiga com as configurações.

```
heroku create nomedoprojeto
```

SUPER IMPORTANTE O NOME DO PROJETO NÃO PODE TER LETRAS MAIUSCULA, NUMEROS OU CARACTERES ESPECIAIS E PRECISA SER UNICO DENTRO DA PLATAFORMA HEROKU

Esse comando serve para criar o seu projeto na Heroku.

```
heroku addons:create heroku-postgresql:hobby-dev
```

Esse comando serve para criar o banco de dados do seu projeto na Heroku.

#10 Passo

Para finalizar só precisamos digitar o seguinte comando, ainda no terminal:

```
git push heroku master
```

No próprio terminal irá aparecer a url que você precisa entrar para abrir o projeto no navegador, mas normalmente a url é https://nomedoprojeto.heroku.com.