



Saiba tudo sobre

Java Web

Versão 1.0

Sumário

Introdução	3
Por que By Casseb?	3
Como funciona a metodologia didática By Casseb?	3
Ciclo básico de Login	4
Pré-requisitos	4
Implantação	4
Criando sua conta no GitHub e baixando template de projeto java web	4
Adquirindo seu link git para uso	7
Preparando ambiente Java em seu pc.....	8
Instalando Eclipse.....	10
Importando o projeto template no Eclipse	10
Instalando plugin do Gradle no eclipse	15
Baixando dependências do template java web pelo gradle	16
Testando a aplicação	17
Utilização.....	19
Tecnologias envolvidas.....	19
Model	19
Criando um objeto Student.java.....	19
Adaptando o objeto na classe Model.java.....	20
Controller	23
Configurando o Controller pela classe REST.java.....	23
Preparando JSON com os dados do Student	23
Preparando o ponto de partida da aplicação – Teste.java	26
View	28
Adaptando nossa página index.html	28
Ciclo Completo	31
Rodando a aplicação	32
Login incorreto	35
Login correto	38

Introdução

Java é uma linguagem de programação muito poderosa, com suporte para execução em diversos sistemas operacionais e sistemas embarcados, Java é muito bem visto para desenvolvimento de soluções complexas e escaláveis, possuindo diversos frameworks e ferramentas voltadas para um uso mais otimizado e rápido.

Pensando na diversidade das ferramentas disponíveis que este documento está sendo proposto, o mesmo será atualizado constantemente, apresentando como extrair o máximo desta fantástica linguagem.

Por que By Casseb?

By Casseb é todo material criado diretamente por Felipe Casseb, fundador da empresa by Casseb, de forma totalmente independente, para fins não-comerciais baseado em seus estudos e conhecimentos na área aplicada.

Como funciona a metodologia didática By Casseb?

Neste documento você terá acesso ao passo a passo relacionado a linguagens, frameworks e boas práticas, sendo declarado quais são os pré-requisitos necessários para o entendimento de cada ferramenta. Não há uma ordem específica para o uso do mesmo, você poderá estudar somente um capítulo, contanto que já atenda aos pré-requisitos do mesmo, sem os pré-requisitos devidamente estudados, não posso garantir um completo entendimento do conteúdo.

Todo capítulo será dividido no máximo em 4 partes:

1. Pré-requisitos – O que você precisa estudar ou comprar antes de iniciar o estudo para aplicar o conteúdo.
2. Implantação – Passo a passo do que precisa ser instalado e configurado para aplicar o conteúdo.
3. Utilização – Explicações sobre o que é, para que serve e como usar cada parte da ferramenta.
4. Práticas – Exemplos práticos utilizando a ferramenta para solução.

Lembrando que não é necessário estudar todas as 4 partes, caso você já tenha domínio da ferramenta e só não sabe implantá-la, pode somente estudar a parte 2 e desenvolver.

Nem sempre 1 capítulo terá as 4 partes, alguns capítulos são somente implantações objetivando capítulos posteriores.

Ciclo básico de Login

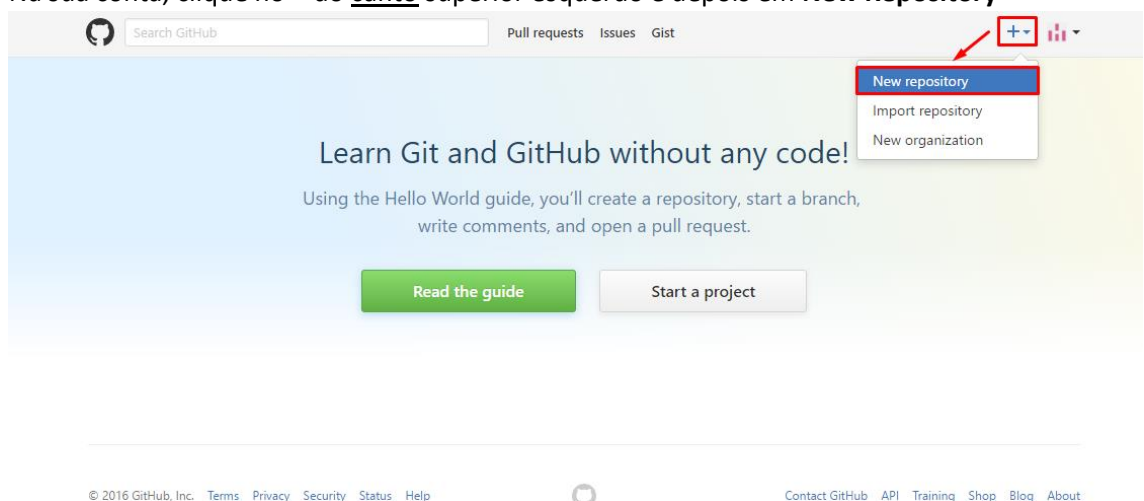
Pré-requisitos

- Mínimo
 - Lógica de programação
 - Java
 - Programação orientada a objetos
- Recomendado
 - DB4O
 - Repositórios de Código
 - Gradle
 - Eclipse
 - Javascript
 - Bootstrap

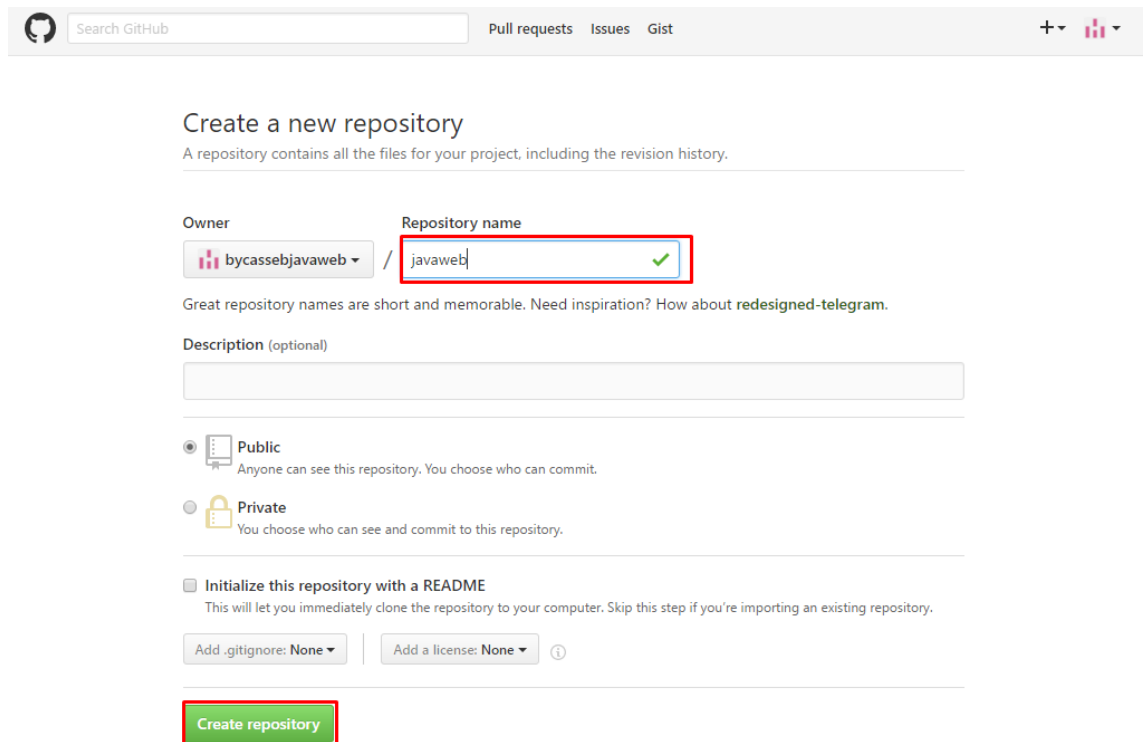
Implantação

Criando sua conta no GitHub e baixando template de projeto java web

- 1- Para dar início, você deve ter uma conta no site github, entre no seguinte link: <https://github.com/join?source=login> e crie sua conta no github.
- 2- Na sua conta, clique no + do canto superior esquerdo e depois em **New Repository**



- 3- Dê um nome para seu repositório, relacionado com o projeto que irá criar e clique em **Create repository**



Search GitHub Pull requests Issues Gist

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: bycassebjavaweb / Repository name: **javaweb** ✓

Great repository names are short and memorable. Need inspiration? How about **redesigned-telegram**.

Description (optional)

☒ **Public**
Anyone can see this repository. You choose who can commit.

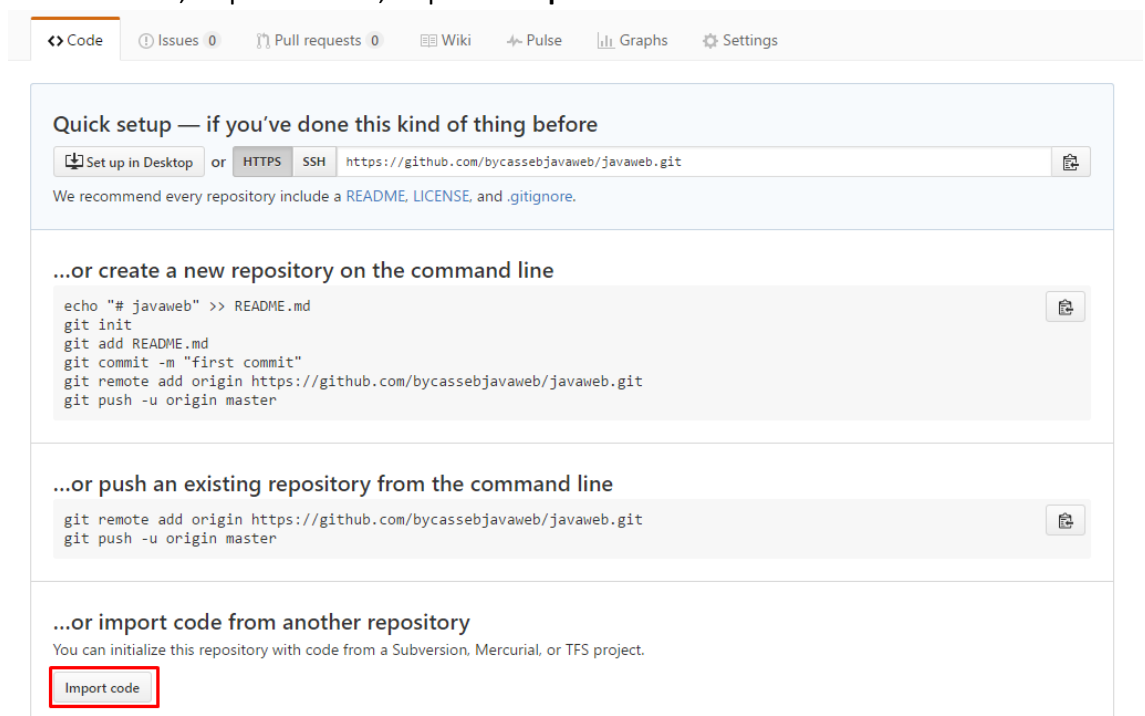
☐ **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None** ⓘ

Create repository

- 4- Nesta etapa, seu repositório está criado, com ele você pode deixar seu código sincronizado na nuvem mantendo-o protegido e fácil para trabalhar com outros colaboradores, na próxima tela, clique em **Import code**.



Code Issues Pull requests Wiki Pulse Graphs Settings

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH `https://github.com/bycassebjavaweb/javaweb.git`

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# javaweb" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/bycassebjavaweb/javaweb.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/bycassebjavaweb/javaweb.git
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

ProTip! Use the URL for this page when adding GitHub as a remote.

- 5- No campo destacado, cole o seguinte caminho:
`https://github.com/casseb/templateJavaWeb.git`
Este link contém um repositório já com uma implantação básica funcionando corretamente e pronta para aplicação no heroku, facilitando bem para este início, depois clique em **Begin import**

Import your project to GitHub


Import all the files, including the revision history, from another version control system.

Your old repository's clone URL

`https://github.com/casseb/templateJavaWeb.git`

Learn more about the types of [supported VCS](#).

Your existing repository

 bycassebjavaweb/javaweb

[Change repository](#)


[Cancel](#)

Begin import

- 6- Após importado seu link do git está pronto para uso para os demais capítulos deste material!

Preparing your new repository

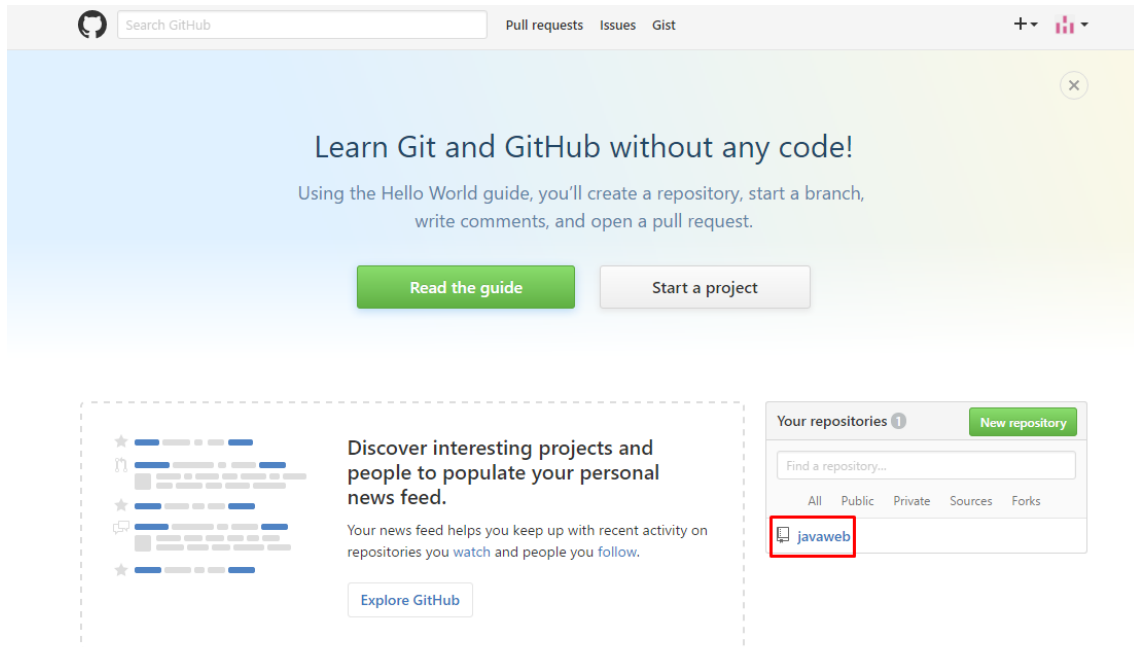
There is no need to keep this window open, we'll email you when the import is done.

 bycassebjavaweb/javaweb

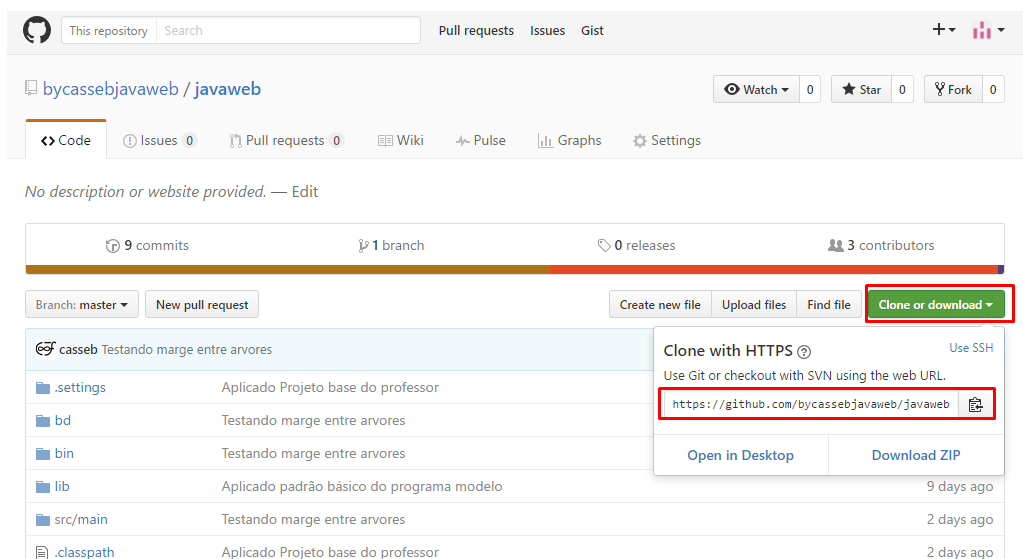
✓ Importing complete! Your new repository [bycassebjavaweb/javaweb](#) is ready.

Adquirindo seu link git para uso

Para recuperar seu link para uso externo, basta entrar em sua conta do github e clicar no projeto que você criou.

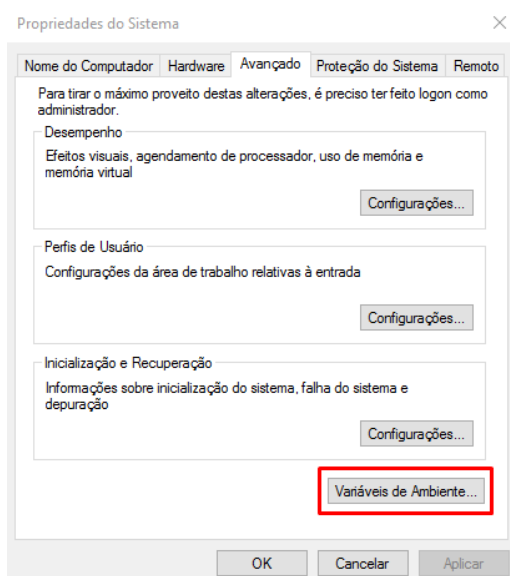
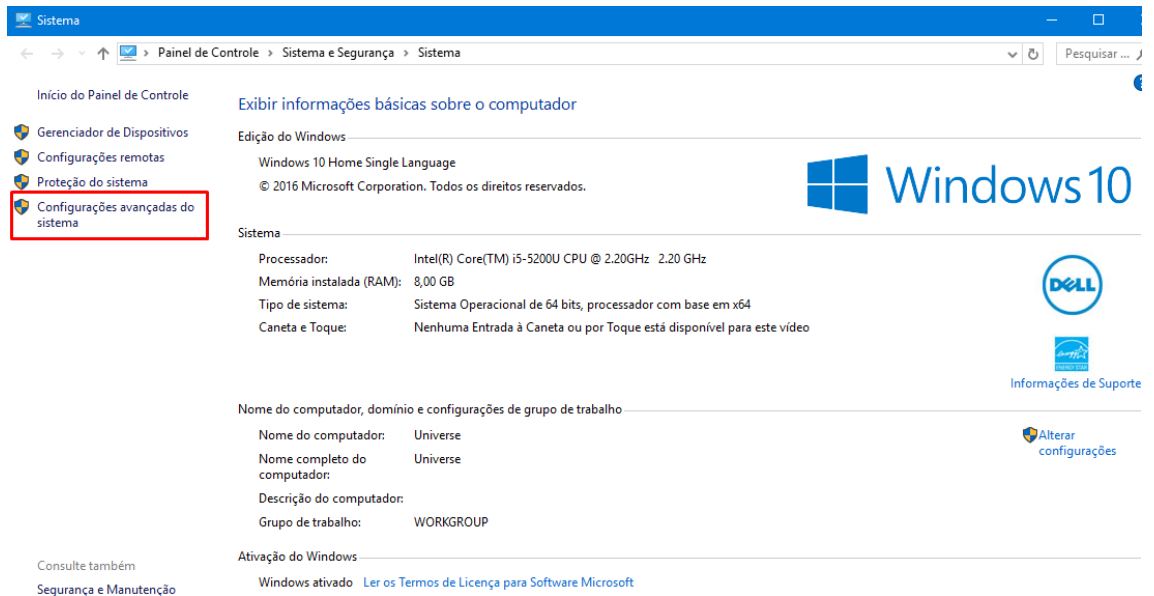


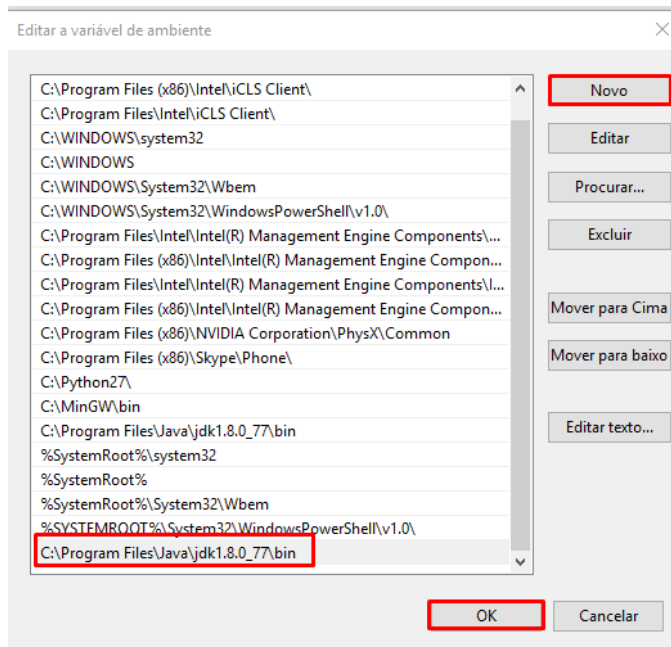
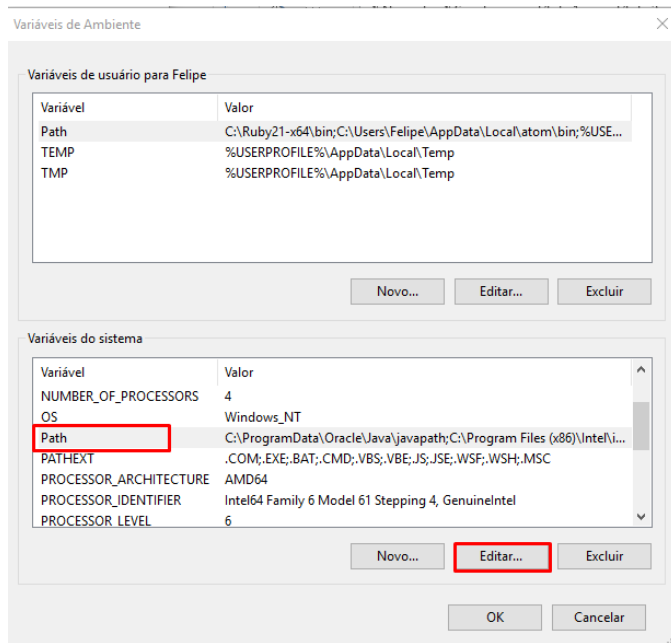
Depois em Clone or Download e o link destaca é o que será usado nos capítulos futuros, neste exemplo o link é: <https://github.com/bycassebjavaweb/javaweb.git>



Preparando ambiente Java em seu pc

- 1- Instale a última versão do jdk em sua máquina pelo seguinte link:
<http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- 2- Após a instalação, localize em sua máquina a pasta onde o Java foi instalado e copie o caminho da pasta **bin** presente na instalação, exemplo:
C:\Program Files\Java\jdk1.8.0_77\bin
- 3- Cole o caminho da sua máquina no path do seu Windows seguindo o seguinte caminho:





- 4- Após este procedimento, entre em seu prompt de comando e digite `java -version`, o resultado deve ser a própria versão instalada, caso seja este o caso a instalação foi bem sucedida.

```
Prompt de Comando
Microsoft Windows [versão 10.0.14393]
(c) 2016 Microsoft Corporation. Todos os direitos reservados.

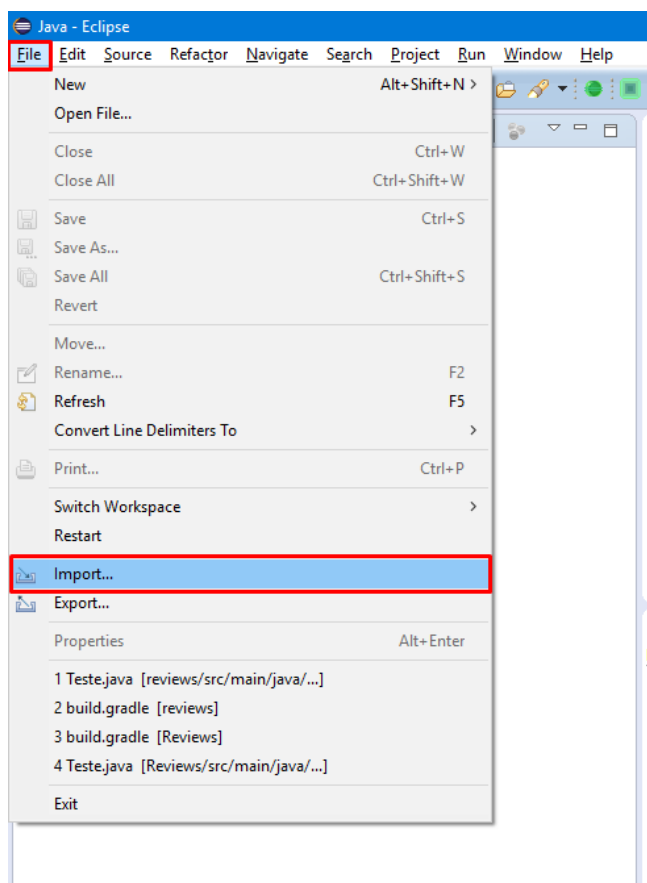
C:\Users\Felipe>java -version
java version "1.8.0_77"
Java(TM) SE Runtime Environment (build 1.8.0_77-b03)
Java HotSpot(TM) 64-Bit Server VM (build 25.77-b03, mixed mode)
```

Instalando Eclipse

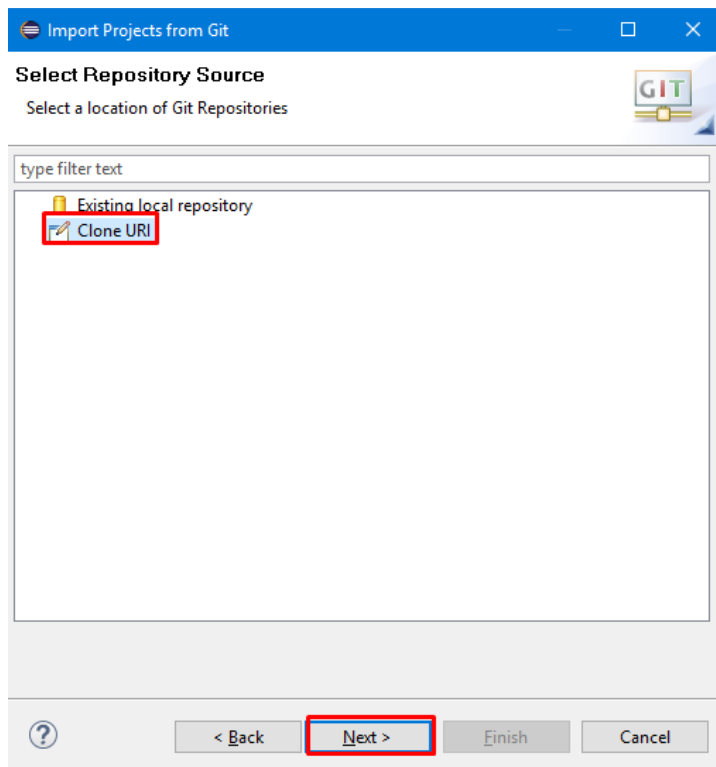
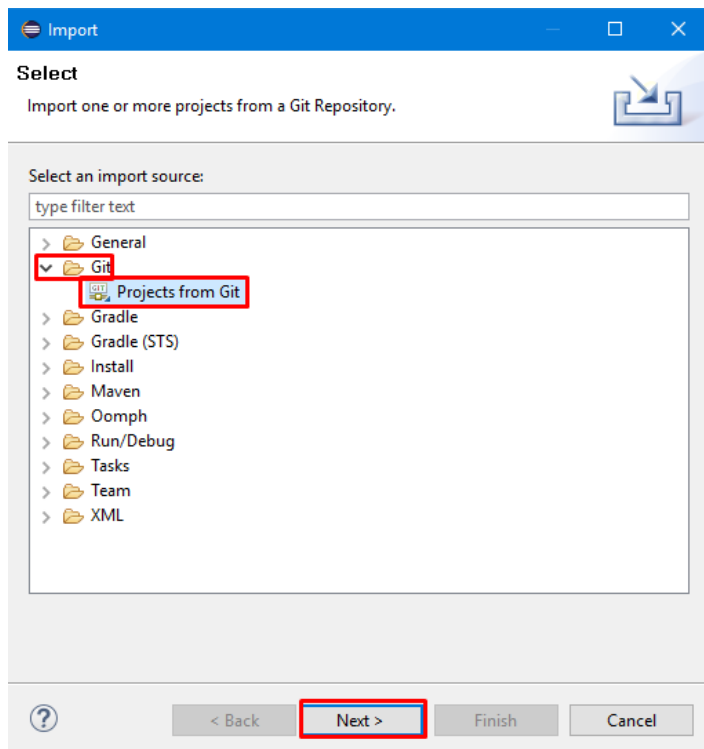
Baixe a versão mais recente do eclipse pelo seguinte link: <https://eclipse.org/downloads/>

Importando o projeto template no Eclipse

- 1- Abra o programa e entre em **File->Import...**



2- Siga os passos das imagens



- 3- Preencha o campo **URI** com o link gerado pelo seu github, depois **user** e **password** da sua conta do github.

Import Projects from Git

Source Git Repository
Enter the location of the source repository.

Location

URI: Local File...

Host:

Repository path:

Connection

Protocol:

Port:

Authentication

User:

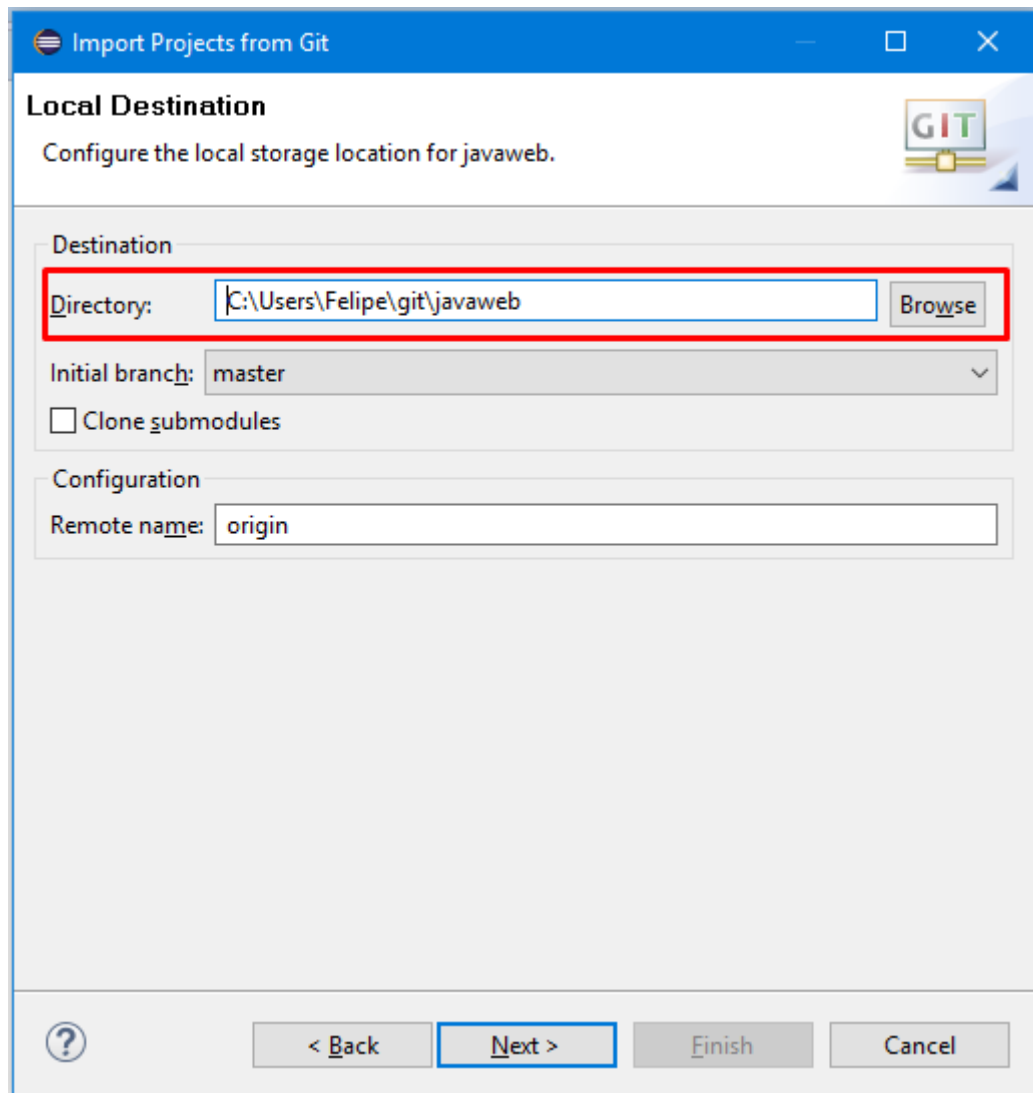
Password:

☐ Store in Secure Store

? < Back Next > Finish Cancel

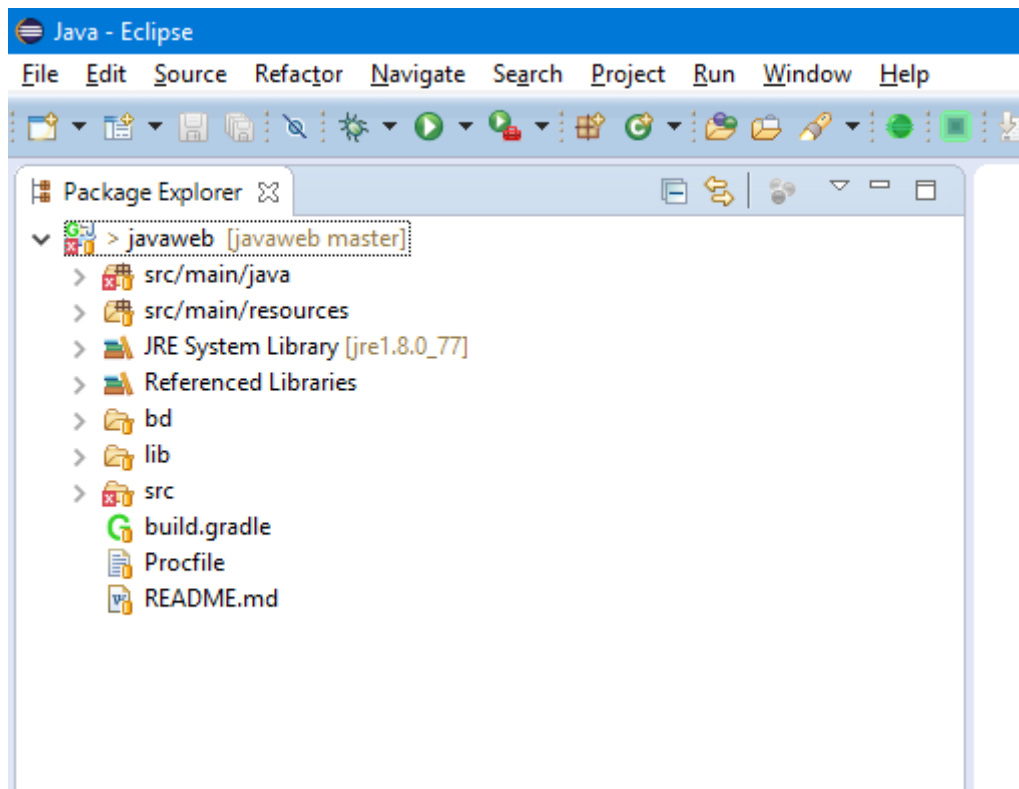
- 4- Deixe selecionado master e clique em next

- 5- Defina no campo Directory onde quer que fique salvo o projeto que você esta baixando:



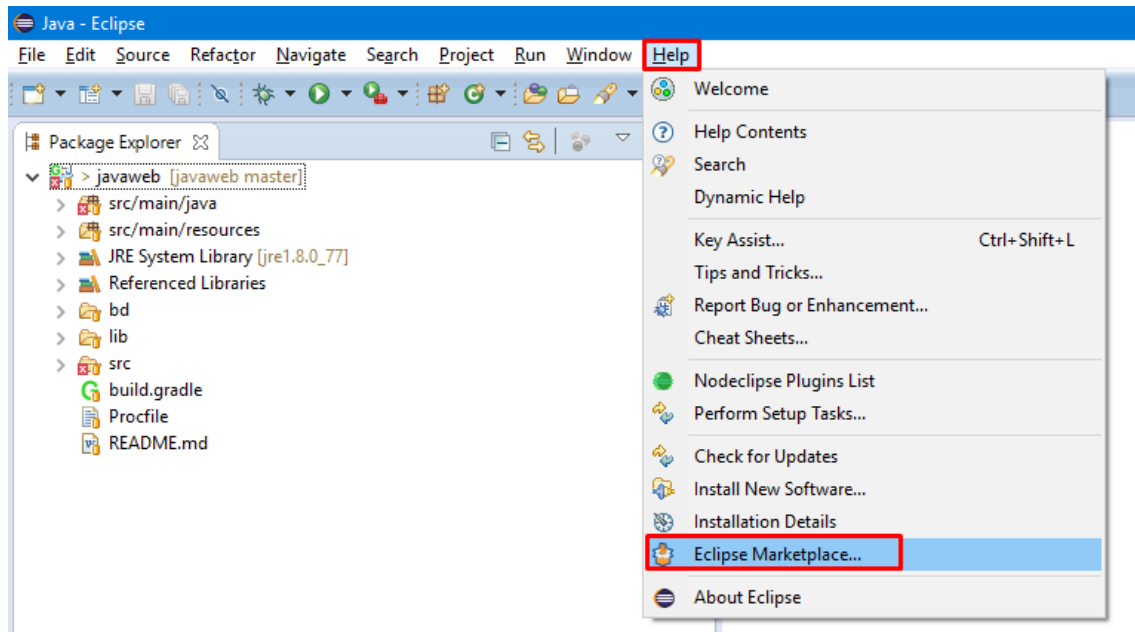
- 6- Vá confirmando até finalizar o wizard.

- 7- Observe que agora o diretório do seu projeto está sendo demonstrado no eclipse e os arquivos em si estão presentes.

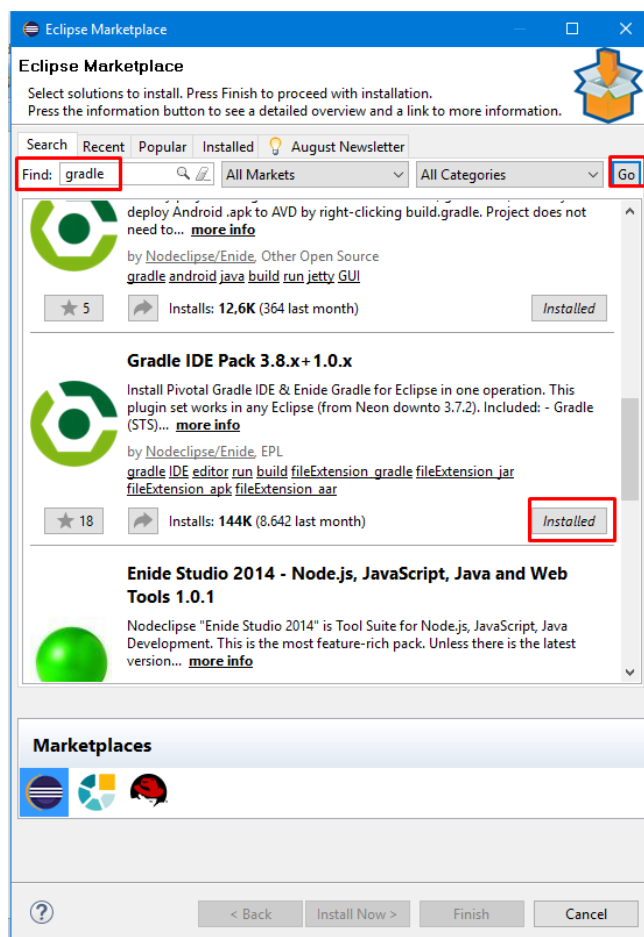


Instalando plugin do Gradle no eclipse

Entre em **help->Eclipse Marketplace**

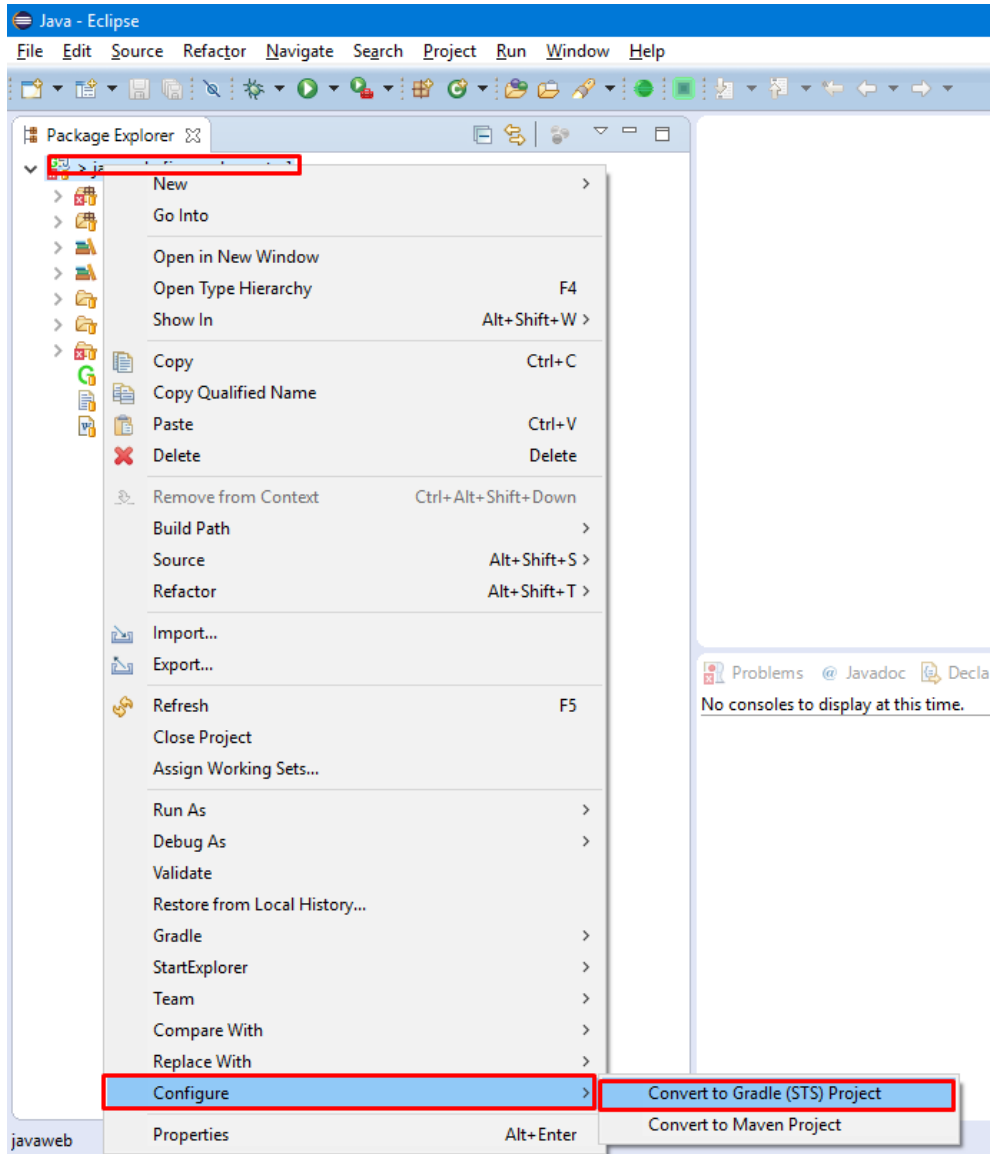


Na tela que abrir localize o gradle e instale



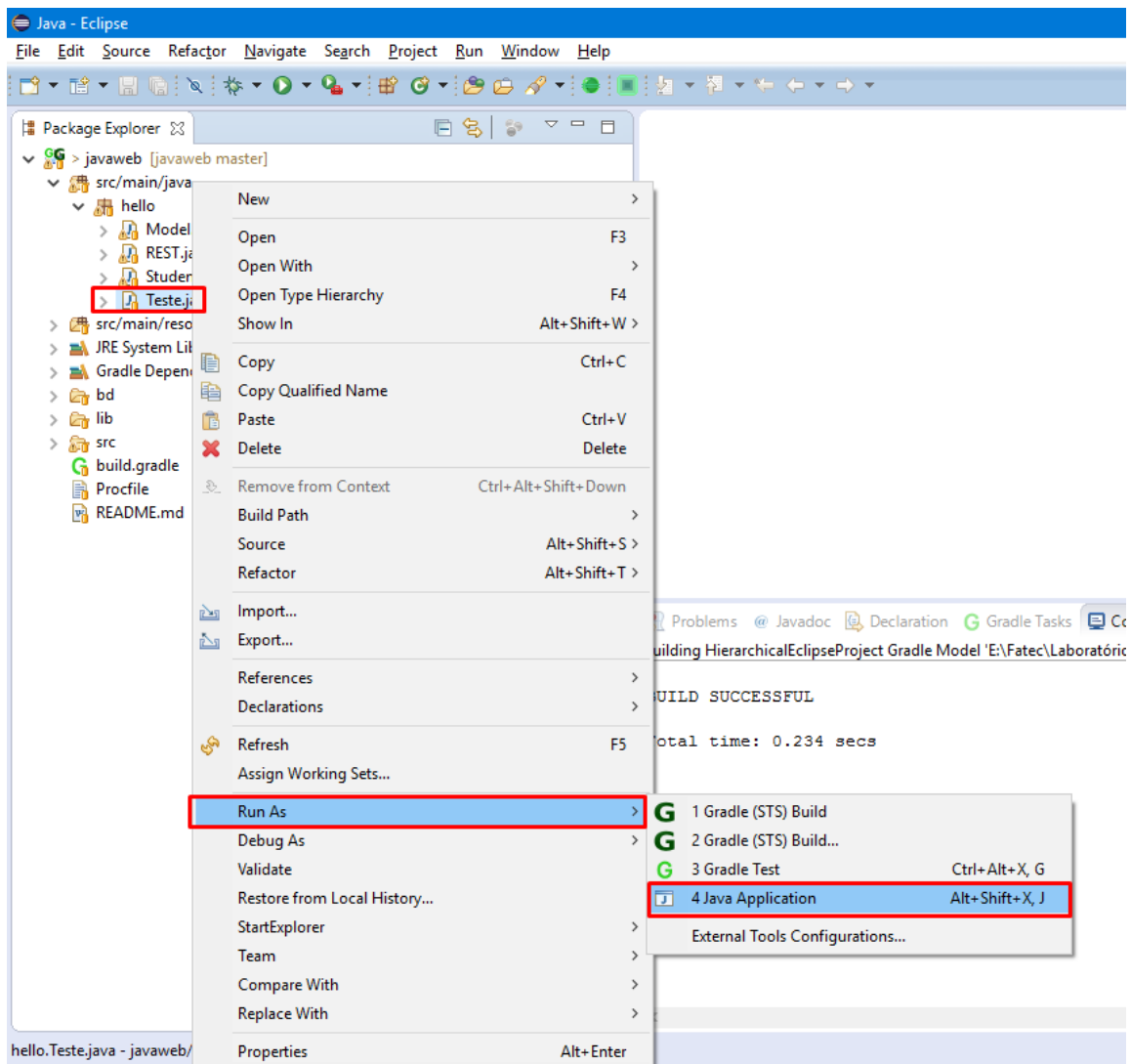
Baixando dependências do template java web pelo gradle

O projeto está com marcações de erro de compilação devida a falta de algumas dependências, graças ao gradle podemos importa-las sem dificuldade, para isto basta clicar com o botão direito no projeto->Configure->Convert to Gradle(STS) Project

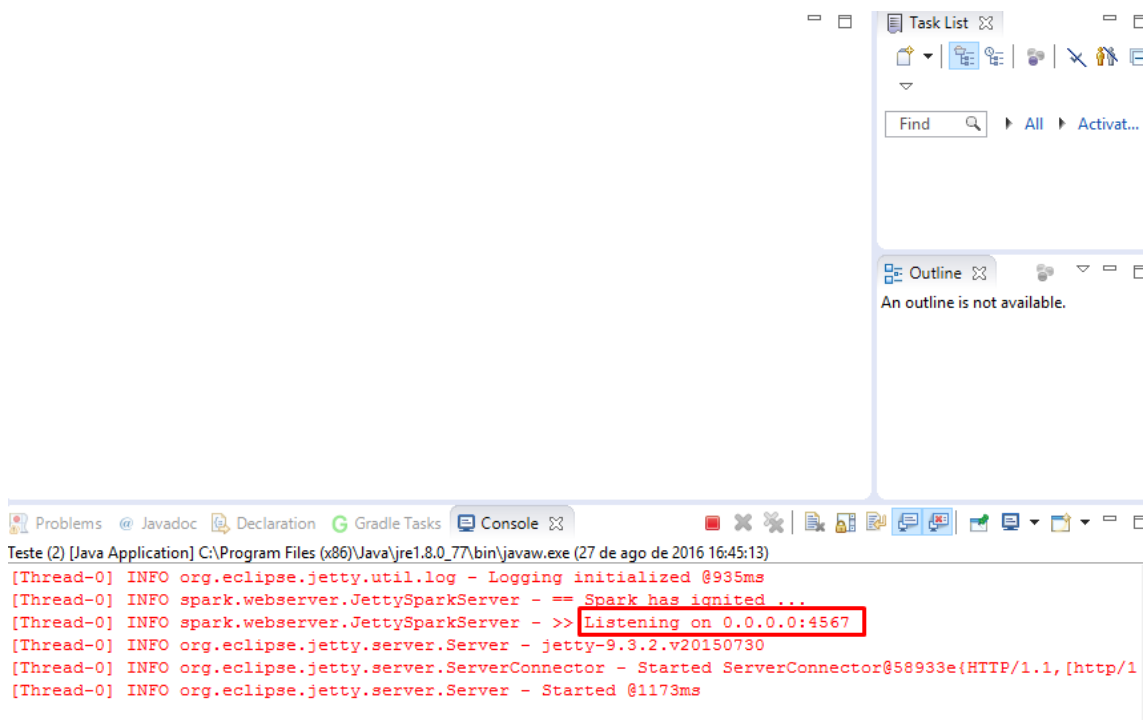


Testando a aplicação

Agora vamos conferir se está executando como deveria, para isso abra o diretório destacado, clique com botão direito em **Teste.java** -> **Run As** -> **Java Application**



Observe que ele te deu um caminho conforme destaque abaixo

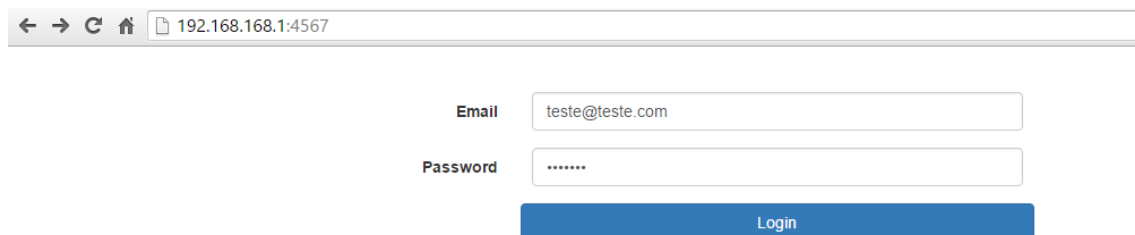


Coloque seu ip seguido da porta destacada, no meu caso é <http://192.168.168.1:4567/>, confira seu ip local pelo prompt de comando digitando **ipconfig**.

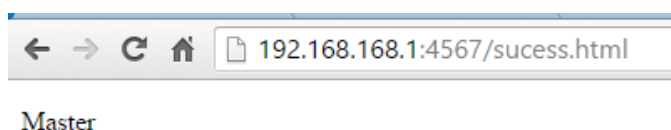
Se abriu uma tela de login, então está compilando corretamente, coloque os seguinte dados:

Email: teste@teste.com

Password: 123456



Caso o resultado ao clicar em **login** seja uma página escrito **Master**, então sua aplicação está executando com sucesso.



Utilização

Tecnologias envolvidas

Nesta aplicação está sendo utilizado os conceitos de MVP (Model View Controller) de forma básica, seu model é composto por banco de dados BD4O que é totalmente orientado a objetos, controller realizado utilizando o protocolo JSON e view utilizando bootstrap e javascript.

Model

Criando um objeto Student.java

Neste projeto você pode criar seu objeto normalmente, utilizando seus conhecimentos de orientação a objeto, definindo seus atributos e métodos como se estivesse utilizando o Java somente no console, como no exemplo abaixo:

```
package hello;

public class Student {

    private String userName;
    private String password;
    private int question;

    public Student(String userName, String password, int question) {
        this.userName = userName;
        this.password = password;
        this.question = question;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public int getQuestion() {
        return question;
    }
    public void setQuestion(int question) {
        this.question = question;
    }
}
```

Adaptando o objeto na classe Model.java

A classe Model foi criada com a finalidade de fazer as consultas, inclusões e até exclusões no banco de dados, trata de toda a persistência de dados de seu sistema.

ObjectContainer

Primeiro criamos um objeto do tipo **ObjectContainer**, este objeto, como o próprio nome diz cria um container de objetos que será utilizado para manipular os dados da aplicação para o banco, quando ele é criado você deve fornecer em qual arquivo de extensão .db4o será persistido, no nosso exemplo o arquivo está presente na pasta bd, tendo como resultado esta linha de código:

```
ObjectContainer students =  
Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(), "bd/students.db4o");
```

Agora o objeto students está vinculado ao arquivo students.db4o para seus dados serem gravados no banco com sucesso.

Adicionando um estudante no banco

Agora vamos criar um método para adicionar um objeto Student no banco, já que ele só vai adicionar, não terá retorno e receberá como atributo um objeto do tipo Student. Será utilizado o método **store** que irá adicionar este objeto ao banco, ficando com esta sintaxe:

```
public void addStudent(Student student) {  
    students.store(student);  
}
```

Consultando um estudante no banco

Crie um objeto do tipo **Query** pois ele tem a finalidade de criar sua consulta, como se você estivesse criando um “select * from”.

Defina uma **constraint** para sua query, neste caso vamos precisar de todos os estudantes, por isso se este método receber Student.class, irá buscar todos os objetos da classe Student do banco.

Crie um objeto do tipo **ObjectSet** para armazenar o resultado da sua consulta, já que a query está pronta, use o método .execute() para executá-la efetivamente.

```
public Student login(String email, String senha){  
    Query query = students.query();  
    query.constrain(Student.class);  
    ObjectSet<Student> allStudents = query.execute();  
}
```

Desta forma o objeto allStudents se tornou uma lista de objetos Student retirados do banco de dados, neste exemplo este método será utilizado para consultar e validar usuário e senha para realizar o login no sistema, retornando o estudante em si, caso o localize e retornando null caso não localize.

```
for (Student student: allStudents){  
    if (student.getUserName().equals(email) &  
        student.getPassword().equals(senha))  
        return student;  
}  
return null;
```

Na próxima página segue o código completo da classe model.java para um melhor entendimento.

```
import com.db4o.Db4oEmbedded;
import com.db4o.ObjectContainer;
import com.db4o.ObjectSet;
import com.db4o.query.Query;

public class Model{

    ObjectContainer students =
    Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(),
    "bd/students.db4o");

    public void addStudent(Student student){
        students.store(student);
    }

    public Student login(String email, String senha){
        Query query = students.query();
        query.constrain(Student.class);
        ObjectSet<Student> allStudents = query.execute();

        for(Student student: allStudents){
            if(student.getUserName().equals(email) &&
student.getPassword().equals(senha)) return student;
        }
        return null;
    }
}
```

Controller

O Controller será responsável por conectar o banco de dados com o visual que o usuário interage.

Configurando o Controller pela classe REST.java

Já que esta classe vai se comunicar com o banco, que está presente no Model.java, vamos criar um atributo do tipo Model para usá-lo na classe, e criar um construtor para que seja criado um REST sempre com um Model vinculado.

```
private Model model;

public REST(Model store) {
    this.model = store;
}
```

Preparando JSON com os dados do Student

Nesta etapa vamos estudar o preparo de um protocolo JSON que irá receber aquele objeto do tipo Student retornado pelo método Login, criado no Model.Java isto, para que o mesmo possa ser utilizado para aplicar o login.

O método getLogin usa um método do tipo get que pode ser utilizado graças ao spark configurado em nosso projeto pelo gradle, nele o **primeiro atributo** define o caminho que, quando chamado, irá retornar no visual esperado.

Se quero uma tratativa ao chamar a url <http://192.168.168.1:4567/login>, tenho que colocar neste parâmetro “/login”, em nosso caso o email e senha do usuário serão direcionados por url para serem tratados, então devemos colocar “/login/:username/:password”, desta forma, caso seja chamado a url <http://192.168.168.1:4567/login/teste/123> será utilizado “teste” e “123” como variáveis que poderão ser usadas dentro da função get.

O **segundo atributo** se refere ao retorno desta comunicação ao browser do usuário, no nosso exemplo foi criado um objeto **Route** e nele foi configurado toda a tratativa de retorno, neste caso o retorno é um objeto Array de JSON, para fixar este entendimento, execute este projeto mandando o seguinte endereço:

<http://localhost:4567/login/teste@teste.com/123456>

Verá que o resultado é um JSON informando o email, caso seja utilizado o seguinte endereço:

<http://localhost:4567/login/teste@teste.com/12345>

O retorno é a chave email com conteúdo vazio, observe que o retorno destas urls são exatamente as tratativas que vamos abordar nas próximas páginas.

```
public void getLogin() {
    get("/login/:username/:password", new Route() {
```

O objeto Route exige a implementação de um método chamado **handle** que cuida exatamente desta etapa de receber os parâmetros mandados por url, o **Request** e **Response** e retornar um **Object**.

```
@Override
public Object handle(final Request request, final Response response) {
```

Iremos agora criar um objeto Student baseado no retorno do nosso método de login, nele vamos passar como parâmetros o usuário e senha da url e, de acordo com a tratativa elabora, ele vai retornar o Student em si caso localize e null caso não localize.

```
try {
    Student student = model.login(request.params(":username"),
    request.params(":password"));
```

Caso este objeto não retorne nulo, sabemos que o usuário existe, por isso vamos criar um objeto JSON com o seguinte padrão: [{"email":"teste@teste.com"}], ou seja, o usuário em si.

```
if(student != null){
    JSONArray jsonResult = new JSONArray();
    JSONObject jsonObj = new JSONObject();
    jsonObj.put("email", student.getUserName()); jsonResult.put(jsonObj);
    return jsonResult;
}
```

Caso o objeto esteja como nulo ele não vai entrar nesta condição, então continuará sua execução gerando um JSON no padrão [{"email":""}], ou seja, email vazio.

```
JSONArray jsonResult = new JSONArray();
JSONObject jsonObj = new JSONObject();
jsonObj.put("email", "");
jsonResult.put(jsonObj);

return jsonResult;
```

Na próxima página deixo o código completo da classe REST.java


```

import static spark.Spark.get;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import spark.Request;
import spark.Response;
import spark.Route;

public class REST{

    private Model model;

    public REST(Model store){
        this.model = store;
    }

    public void getLogin(){

        get("/login/:username/:password", new Route() {
            @Override
            public Object handle(final Request request, final Response response){

                response.header("Access-Control-Allow-Origin", "*");

                try {
                    Student student =
model.login(request.params(":username"), request.params(":password"));

                    if(student != null){
                        JSONArray jsonResult = new JSONArray();
                        JSONObject jsonObj = new JSONObject();
                        jsonObj.put("email", student.getUserName());
                        jsonResult.put(jsonObj);
                        return jsonResult;
                    }
                }
                catch (JSONException e) {
                    //e.printStackTrace();
                }

                JSONArray jsonResult = new JSONArray();
                JSONObject jsonObj = new JSONObject();
                jsonObj.put("email", "");
                jsonResult.put(jsonObj);

                return jsonResult;
            }
        });
    }
}

```

Preparando o ponto de partida da aplicação – Teste.java

Agora na classe Teste.java iniciamos criando um atributo do tipo Model, para que haja a comunicação com o Model.java e criar a classe main (para que seja iniciado a aplicação a partir dela)

```
final static Model model = new Model();

public static void main(String[] args) {
```

Criamos um objeto do tipo **ProcessBuilder** que tem como finalidade criar um processo no sistema operacional, um objeto **Integer** para armazenar o número da porta que deseja utilizar, caso a porta já esteja sendo utilizada pelo ambiente, ele continua com o mesmo, caso contrário pega o definido no código, no nosso exemplo a porta é 4567, depois de alimentado a variável port, a mesma é usada na função port que define a porta que será usada.

```
ProcessBuilder process = new ProcessBuilder();
Integer port;
if (process.environment().get("PORT") != null) {
    port = Integer.parseInt(process.environment().get("PORT"));
} else {
    port = 4567;
}
port(port);
```

Criamos a função **inicialize()** que terá como objetivo criar um usuário em tempo de execução e adiciona-lo ao banco usando aquela função **addStudent** explicado anteriormente, desta forma podemos realizar nosso teste sem o risco do não ter um usuário e senha no banco.

```
public static void initialize () {
    model.addStudent(new Student("teste@teste.com", "123456", 0));
}
```

Definimos onde está localizado nossos arquivos estáticos, ou seja, arquivos que não são .java, neste caso as views (html, css, javascript) pelo spark

```
staticFileLocation("/static");
```

Por último instanciamos uma classe REST passando nosso Model criado acima e executamos a função **getLogin** para iniciar aquela tratativa de urls que foi tratado no REST.

```
REST controller = new REST(model);
controller.getLogin();
```

Código completo abaixo da classe Teste.java para melhor entendimento:

```
import static spark.Spark.*;

public class Teste {

    final static Model model = new Model();

    public static void main(String[] args) {

        // Get port config of heroku on environment variable
        ProcessBuilder process = new ProcessBuilder();
        Integer port;
        if (process.environment().get("PORT") != null) {
            port = Integer.parseInt(process.environment().get("PORT"));
        } else {
            port = 4567;
        }
        port(port);

        initialize();

        staticFileLocation("/static");
        REST controller = new REST(model);
        controller.getLogin();
    }

    public static void initialize () {
        model.addStudent(new Student("teste@teste.com", "123456", 0));
    }
}
```

View

Será explicado a comunicação entre a interface direta e o seu respectivo Controller, assim ficará claro como uma página estática html irá enviar e receber informações para o banco.

[Adaptando nossa página index.html](#)

Em nosso formulário deve-se atentar sobre os atributos id, eles que relacionam o componente html com o javascript da página, no nosso exemplo, temos id="form" para a tag que comporta todo o formulário, id="username" e id="password" em seus respectivos inputs. O botão que irá utilizar o javascript deve ser do tipo "submit", para que dispare a função javascript relacionada.

```
<form class="form-horizontal" method="post" id="form">
  <div class="form-group">
    <label for="inputEmail3" class="col-md-4 control-label">Email</label>
    <div class="col-md-4">
      <input required="required" type="email"
class="form-control" id="username" placeholder="Email">
    </div>
  </div>
  <div class="form-group">
    <label for="inputPassword3" class="col-md-4 control-label">Password</label>
    <div class="col-md-4">
      <input required="required" type="password"
class="form-control" id="password" placeholder="Password">
    </div>
  </div>
  <button id="login" type="submit" class="btn btn-primary col-md-offset-4 col-md-4">Login</button>
</form>
```

A função javascript deve seguir a sintaxe abaixo, já que na segunda linha é chamado \$('form').submit, o que esta dentro da função irá executar ao usuário clicar no botão submit presente no formulário form.

```
$(document).ready(function() {
  $('form').submit(function(e) {
```

Depois declaramos duas variáveis para receber usuário e senha do formulário.

```
var username = $('#username').val().trim();
var password = $('#password').val().trim();
```

Utilizamos em seguida uma função que captura um JSON de uma determinada url e alimenta uma variável nova, que no nosso exemplo é data.

```
$.getJSON("/login/"+username+"/"+password+"?format=json&jsoncallback=", function(data) {
```

Confira que o formato chamado é o mesmo que o spark aceita para consultar no banco, então esta chamada vai mandar via url o usuário e senha e retornar uma lista de JSON que encontrar.

Em sua próxima condição, ele confere se o retorno foi email == "", se for este o caso ele redireciona para a página index.html, se é diferente de vazio ele direciona para a página sucess.html, que é a página que abre quando conseguimos logar.

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Login</title>
    <script src="http://code.jquery.com/jquery-
latest.min.js"></script>
    <link href="http://getbootstrap.com/dist/css/bootstrap.min.css"
rel=stylesheet>

    <script>
      $(document).ready(function() {
        $('form').submit(function(e) {
          e.preventDefault();
          var username = $('#username').val().trim();
          var password = $('#password').val().trim();
          $.getJSON("/login/"+username+"/"+password+"?format=json&jsoncal
lback=",function(data) {
            if(data[0].email == ""){

              window.location.href =

'/index.html';

            } else {

              window.location.href =

'/sucess.html';

            }
          });
        });
      });
    </script>
  </head>
  <body>
    <form class="form-horizontal" method="post" id="form">
      <div class="form-group">
        <label for="inputEmail3" class="col-md-4
control-label">Email</label>
        <div class="col-md-4">
          <input required="required" type="email"
class="form-control" id="username" placeholder="Email">
        </div>
      </div>
      <div class="form-group">
        <label for="inputPassword3" class="col-md-4
control-label">Password</label>
        <div class="col-md-4">
          <input required="required"
type="password" class="form-control" id="password"
placeholder="Password">
        </div>
      </div>
      <button id="login" type="submit" class="btn btn-
primary col-md-offset-4 col-md-4">Login</button>
    </form>
  </body>
</html>

```

Ciclo Completo

Para finalizar o conteúdo deste capítulo irei explicar por imagens o ciclo completo do início da execução do programa, o que acontece quando você digita usuário e senha correto e quando digita incorreto.

Rodando a aplicação

```
Teste.java ✕
1 package hello;
2
3 import static spark.Spark.*;
4
5 public class Teste { Instanciando a classe Model
6
7     final static Model model = new Model();
8
9     public static void main(String[] args) {
10
```

```
Model.java ✕
1 package hello;
2
3 import com.db4o.Db4oEmbedded;
4
5 public class Model {
6
7     // Prepara um container para armazenar Student
8     ObjectContainer students = Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(), "bd/students.db4o");
9
10
11
12
```

```
Teste.java ✕
1 package hello;
2
3 import static spark.Spark.*;
4
5 public class Teste {
6
7     final static Model model = new Model();
8
9     public static void main(String[] args) {
10
11         // Get port config of heroku on environment variable
12         ProcessBuilder process = new ProcessBuilder();
13         Integer port;
14         if (process.environment().get("PORT") != null) {
15             port = Integer.parseInt(process.environment().get("PORT"));
16         } else {
17             port = 4567;
18         }
19         port(port);
20
```

Iniciado o Spark reservando a porta 4567 para uso


```

1 package hello;
2
3 import static spark.Spark.*;
4
5 public class Teste {
6
7     final static Model model = new Model();
8
9     public static void main(String[] args) {
10
11         // Get port config of heroku on environment variable
12         ProcessBuilder process = new ProcessBuilder();
13         Integer port;
14         if (process.environment().get("PORT") != null) {
15             port = Integer.parseInt(process.environment().get("PORT"));
16         } else {
17             port = 4567;
18         }
19         port(port);
20
21         initialize();
22
23         staticFileLocation("/static");
24         REST controller = new REST(model);
25         controller.getLogin();
26     }
27
28     public static void initialize () {
29         model.addStudent(new Student("teste@teste.com", "123456", 0));
30     }
31 }

```

Executa método initialize que irá adicionar um estudante ao model instanciado

```

1 package hello;
2
3 import com.db4o.Db4oEmbedded;
4
5 public class Model {
6
7     ObjectContainer students = Db4oEmbedded.openFile(Db4oEmbedded.
8
9
10
11
12
13     public void addStudent(Student student) {
14         students.store(student);
15     }
16

```

Agora o Student teste@teste.com esta no banco

```

1 package hello;
2
3 import static spark.Spark.*;
4
5 public class Teste {
6
7     final static Model model = new Model();
8
9     public static void main(String[] args) {
10
11         // Get port config of heroku on environment variable
12         ProcessBuilder process = new ProcessBuilder();
13         Integer port;
14         if (process.environment().get("PORT") != null) {
15             port = Integer.parseInt(process.environment().get("PORT"));
16         } else {
17             port = 4567;
18         }
19         port(port);
20         initialize();
21
22         staticFileLocation("/static");
23         REST controller = new REST(model);
24         controller.getLogin();
25     }
26 }

```

Foi definido /static como localização dos arquivos estáticos
criado uma instância de REST recebendo o model e
executado o método getLogin

```

public class REST{
    private Model model;
    public REST(Model store){
        this.model = store;
    }
}

```

Ao chamar o construtor, REST
passa a ter acesso ao Model

```

public void getLogin(){
    get("/login/:username/:password", new Route() {
        @Override
        public Object handle(final Request request, final Response response){

            response.header("Access-Control-Allow-Origin", "*");

            try {
                Student student = model.login(request.params(":username"), request.params(":password"));

                if(student != null){
                    JSONArray jsonResult = new JSONArray();
                    JSONObject jsonObj = new JSONObject();
                    jsonObj.put("email", student.getUserName());
                    jsonResult.put(jsonObj);
                    return jsonResult;
                }
            } catch (JSONException e) {
                //e.printStackTrace();
            }

            JSONArray jsonResult = new JSONArray();
            JSONObject jsonObj = new JSONObject();
            jsonObj.put("email", "");
            jsonResult.put(jsonObj);

            return jsonResult;
        }
    });
}

```

GetLogin foi executado, ficando pronto para ser usado quando
o padrão /login/username/password for acessado

Login incorreto

Email

Password

Login

Digitado usuário não cadastrado e clicado em Login

```

<script>
$(document).ready(function() {
    $('form').submit(function(e) {

        e.preventDefault();

        var username = $('#username').val().trim();
        var password = $('#password').val().trim();

        $.getJSON("/login/"+username+"/"+password+"?format=json&jsoncallback=",function(data) {

            if(data[0].email == ""){

                window.location.href = '/index.html';

            } else {

                window.location.href = '/sucess.html';

            }

        });

    });
});
</script>
</head>

```

No index.html, username recebe "teste@testeerrado.com" e senha "123456" acessando a seguinte url:
http://localhost:4567/login/teste@testeerrado.com/123456

```

public void getLogin(){

    get("/login/:username/:password", new Route() {
        @Override
        public Object handle(final Request request, final Response response){

            response.header("Access-Control-Allow-Origin", "*"); teste@testeerrado.com

            try {
                Student student = model.login(request.params(":username"), request.params(":password")); 123456

                if(student != null){
                    JSONArray jsonResult = new JSONArray();
                    JSONObject jsonObj = new JSONObject();
                    jsonObj.put("email", student.getUserName());
                    jsonResult.put(jsonObj);
                    return jsonResult;
                }
            }
            catch (JSONException e) {
                //e.printStackTrace();
            }
        }
    });
}

```

A url executa a função getLogin da classe REST.java, que alimenta uma variável do tipo Student com um retorno da função login da classe Model.java que recebe os parametros da url

```

public Student login(String email, String senha){
    Query query = students.query();
    query.constrain(Student.class);
    ObjectSet<Student> allStudents = query.execute();

    for(Student student: allStudents){
        if(student.getUserName().equals(email) && student.getPassword().equals(senha)) return student;
    }
    return null;
}

```

O método login presente em Model.java não localiza o usuário e retorna null

```

public void getLogin(){

    get("/login/:username/:password", new Route() {
        @Override
        public Object handle(final Request request, final Response response){

            response.header("Access-Control-Allow-Origin", "*");

            try {
                Student student = model.login(request.params(":username"), request.params(":password"));

                if(student != null){
                    JSONArray jsonResult = new JSONArray();
                    JSONObject jsonObj = new JSONObject();
                    jsonObj.put("email", student.getUserName());
                    jsonResult.put(jsonObj);
                    return jsonResult;
                }
            } catch (JSONException e) {
                //e.printStackTrace();
            }

            JSONArray jsonResult = new JSONArray();
            JSONObject jsonObj = new JSONObject();
            jsonObj.put("email", "");
            jsonResult.put(jsonObj);

            return jsonResult;
        }
    });
}

```

Com um retorno nulo, o método getLogin da classe REST retorna um JSON com usuário definido por ""

```

index.html
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <title>Login</title>
6
7     <script src="http://code.jquery.com/jquery-latest.min.js"></script>
8
9     <link href="http://getbootstrap.com/dist/css/bootstrap.min.css" rel="stylesheet">
10
11     <script>
12       $(document).ready(function() {
13         $('#form').submit(function(e) {
14
15           e.preventDefault();
16
17           var username = $('#username').val().trim();
18           var password = $('#password').val().trim();
19
20           $.getJSON("/login/"+username+"/"+password+"?format=json&jsoncallback=", function(data) {
21
22             if (data[0].email == "") {
23               window.location.href = '/index.html';
24             } else {
25
26               window.location.href = '/sucess.html';
27
28             }
29           });
30         });
31       });
32     });
33

```

index.html irá receber um JSON que email == "" pois isto direciona o usuario para /index.html

← → ↺ 🏠 localhost:4567/index.html

Página redirecionada novamente para o início

Email

Password

Login

Login correto

Email

Password

Login

Digitado usuário e senha existente e clicado em Login

```

1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <title>Login</title>
6
7     <script src="http://code.jquery.com/jquery-latest.min.js"></script>
8
9     <link href="http://getbootstrap.com/dist/css/bootstrap.min.css" rel="stylesheet">
10
11   <script>
12     $(document).ready(function() {
13       $('form').submit(function(e) {
14
15         e.preventDefault();
16
17         var username = $('#username').val().trim();
18         var password = $('#password').val().trim();
19
20         $.getJSON("/login/"+username+"/"+password+"?format=json&jsoncallback=", function(data) {
21
22           if(data[0].email == "") {
23
24             window.location.href = '/index.html';
25
26           } else {
27
28             window.location.href = '/sucess.html';
29
30           }
31         });
32       });
33     });
34   </script>

```

A função javascript de index.html gera a url
<http://localhost:4567/login/teste@teste.com/123456>

```

public void getLogin(){

    get("/login/:username/:password", new Route() {
        @Override
        public Object handle(final Request request, final Response response){

            response.header("Access-Control-Allow-Origin", "*");

            try {
                Student student = model.login(request.params(":username"), request.params(":password"));

                if(student != null){
                    JSONArray jsonResult = new JSONArray();
                    JSONObject jsonObj = new JSONObject();
                    jsonObj.put("email", student.getUserName());
                    jsonResult.put(jsonObj);
                    return jsonResult;
                }
            }
            catch (JSONException e) {
                //e.printStackTrace();
            }
        }
    });
}

```

Dispara o método getLogin da classe REST.java onde usa o método login da classe Model.java para conferir se o usuário existe

```

public Student login(String email, String senha){
    Query query = students.query();
    query.constrain(Student.class);
    ObjectSet<Student> allStudents = query.execute();

    for(Student student: allStudents){
        if(student.getUserName().equals(email) && student.getPassword().equals(senha)) return student;
    }
    return null;
}

```

Aqui ele encontra o usuário, pois isso retorna o objeto Student relacionado com usuário e senha digitados.

```

get("/login/:username/:password", new Route() {
    @Override
    public Object handle(final Request request, final Response response){

        response.header("Access-Control-Allow-Origin", "*");

        try {
            Student student = model.login(request.params(":username"), request.params(":password"));

            if(student != null){
                JSONArray jsonResult = new JSONArray();
                JSONObject jsonObj = new JSONObject();
                jsonObj.put("email", student.getUserName());
                jsonResult.put(jsonObj);
                return jsonResult;
            }
        }
        catch (JSONException e) {
            //e.printStackTrace();
        }
    }
});

```

Agora que REST.java recebeu um objeto Student ele prepara o retorno de um JSON com o respectivo e-mail do usuário

```
<script>
    $(document).ready(function() {
        $('form').submit(function(e) {

            e.preventDefault();

            var username = $('#username').val().trim();
            var password = $('#password').val().trim();

            $.getJSON("/login/"+username+"/"+password+"?format=json&jsoncallback=", function(data) {

                if(data[0].email == "") {

                    window.location.href = '/index.html'; javascript redireciona a página para sucess.html

                } else {

                    window.location.href = '/sucess.html';

                }

            });

        });
    });
</script>
```

