

Programação Funcional

Unidade 10 – Criação de API Rest com Clojure

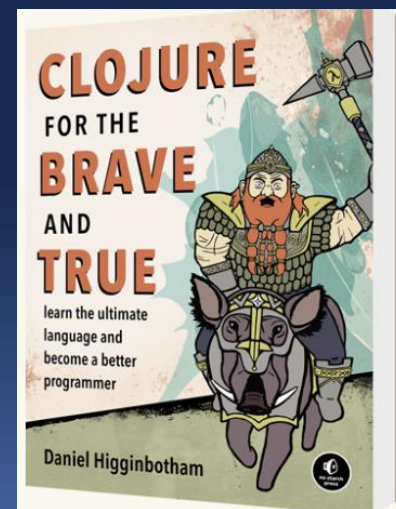
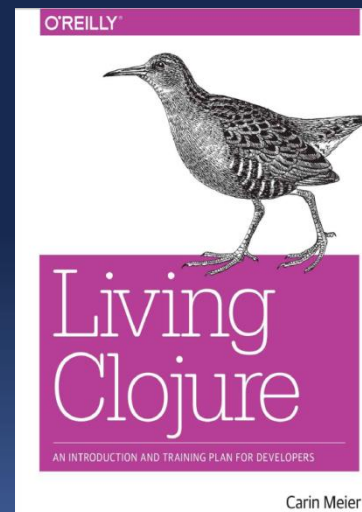
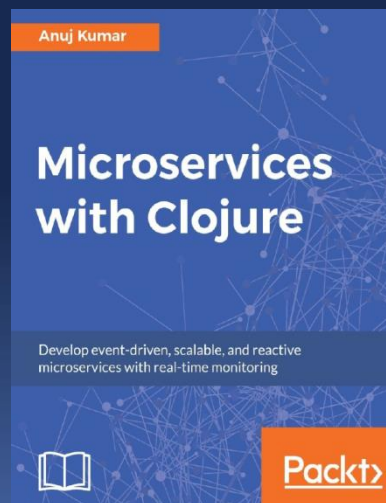
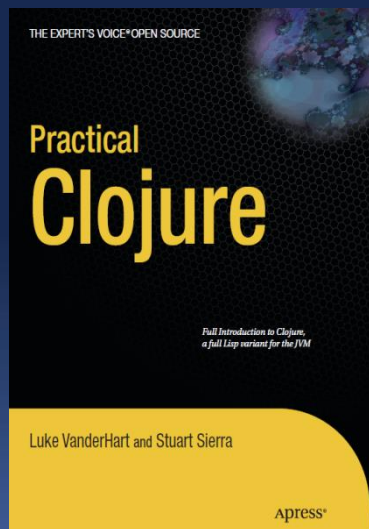
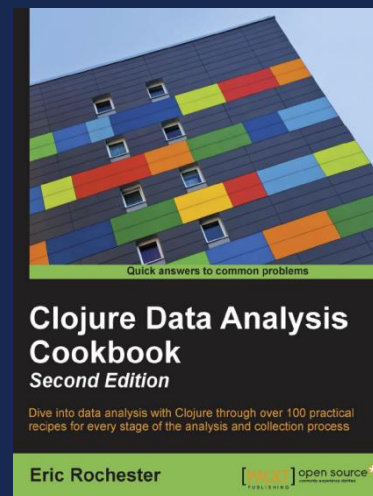
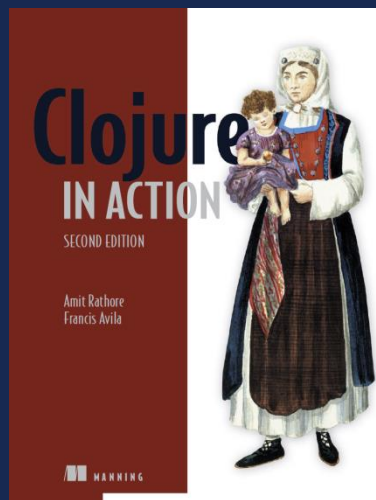
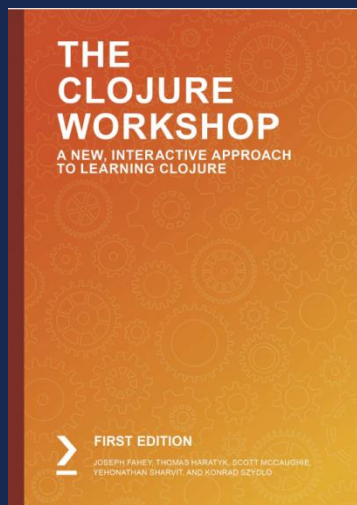


Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecido.freitas@prof.uscs.edu.br
aparecidovfreitas@gmail.com



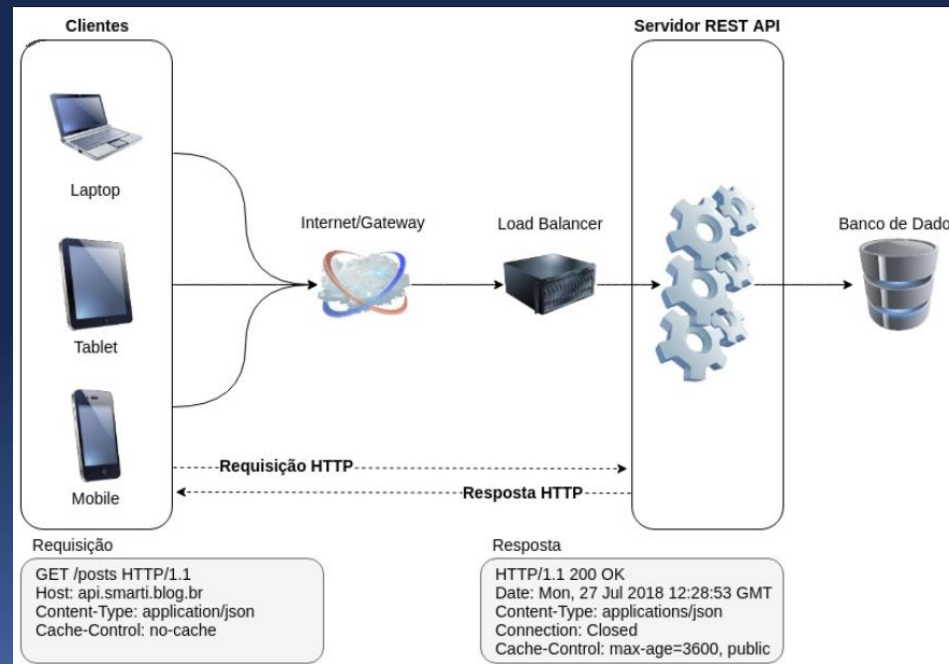
Revisão Técnica: Mauricio Szabo
mauricio.szabo@gmail.com

Bibliografia



API's Rest

- Uma **API** (Application Programming Interface) permite a comunicação entre dois sistemas;
- Uma **API** fornece essencialmente a linguagem e o contrato para a interação entre dois sistemas;
- Cada **API** tem a especificação que determina a forma pela qual as informações podem ser transferidas;



REST

- **REST** é uma abreviação de "**Representational State Transfer**" ou transferência de estado representacional;
- Trata-se de um modelo de arquitetura para sistemas distribuídos;
- O modelo foi criado por **Roy Fielding**, um dos criadores do protocolo **HTTP**;
- A ideia por trás desse modelo é viabilizar a transferência de dados via rede.



REST

- **REST** tem sido a base para a evolução do protocolo **HTTP**;
- Tem sido usado como alternativa ao modelo **SOAP** o qual é baseado em **XML**;
- Atualmente, graças à sua flexibilidade, tem sido largamente utilizado na construção de **API's** para integração entre sistemas **Web**;



Recursos

- Toda aplicação Web gerencia **recursos**;
- Suponha que iremos desenvolver uma **API Rest** que manipula cursos;
- À medida em que formos desenvolvendo novas funcionalidades, novos recursos também estarão presentes na aplicação, como por exemplo, disciplinas, professores, grades de curso, etc
- Assim, precisamos de alguma forma **diferenciar** o manuseio de um recurso de outro. Por exemplo, poderíamos ter uma API que retornasse os cursos, outra que retornasse os professores, e assim por diante.
- Para isso, definimos um identificador único para recurso a ser retornado. Esse identificador é chamado URI. Por exemplo, para retornar cursos a URI poderia ser **/cursos**, para professores poderia ser **/professores**, e assim por diante.



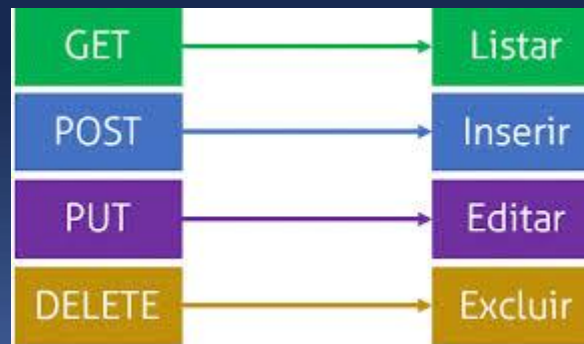


Ok ! Cada recurso está associado a uma URI !

Mas, como especificar o que deve ser feito com o recurso ?

Operações com Recursos

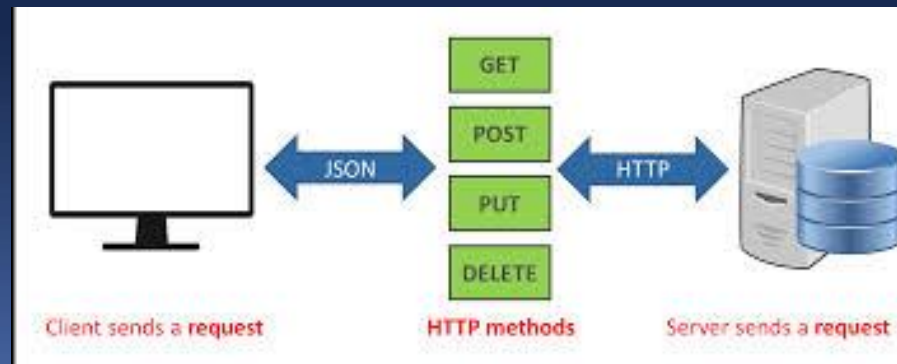
- Imaginemos que a URI **/cursos** está associada ao recurso **"cursos"** ;
- Mas, como especificar o que será feito com o recurso **"cursos"** ?
- Pode haver **diferentes** formas de se tratar o recurso **"cursos"**;
- Por exemplo, pode-se **retornar** uma lista completa dos cursos, pode-se **atualizar** o recurso cursos, pode-se **excluir** o recurso, etc;
- Assim, para se definir a operação a ser utilizada com o recurso, utilizam-se os **verbos HTTP** ou os métodos **HTTP**;
- Por exemplo: **GET** para se recuperar recursos, **POST** para se cadastrar recursos, **PUT** para atualizar ou ainda **DELETE** para se excluir.



- GET /alunos
- POST /alunos
- PUT /alunos/{id}
- DELETE /alunos/{id}

API's Rest

- Da mesma forma que uma página **Web** é renderizada pelo Browser, as API's podem usar requisições HTTP para obter informações de uma aplicação ou servidor web;
- API's Rest oferecem uma forma mais **leve**, usando URL's na maioria dos casos, para enviar e receber informações;
- Rest usa quatro verbos HTTP 1.1 diferentes (**GET**, **POST**, **PUT** e **DELETE**) para executar tarefas;
- Os recursos a serem manipulados são representados em um determinado **formato** (Json, XML, texto, etc...), daí o nome **REST**, onde **R** = Representational.



Representação do recurso

- Embora **Json** tem sido muito usado, outras formas de se representar os dados é possível.

```
{
  "employee": "Max Mustermann",
  "items": [
    {
      "name": "TestData",
      "quantity": "2"
    },
    {
      "name": "Test2",
      "quantity": "3"
    },
    {
      "name": "Test3",
      "quantity": "2"
    }
  ],
  "table": "Tisch 5"
}
```

JSON

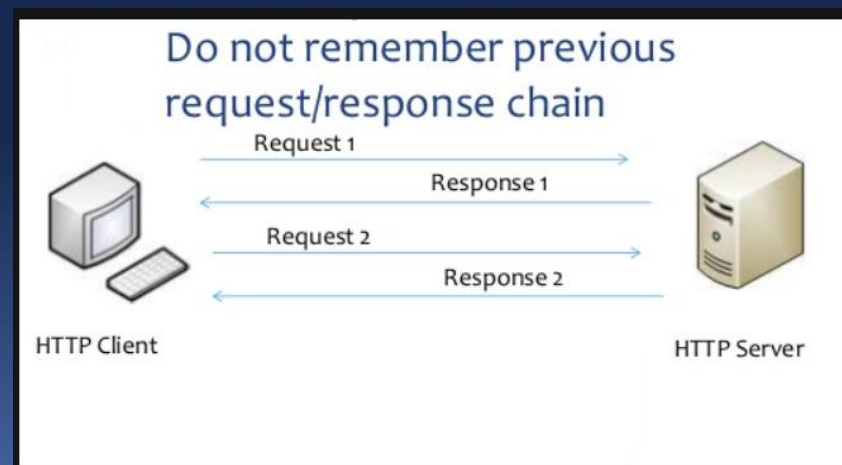
```
<?xml version="1.0"?>
<quiz>
  <qanda seq="1">
    <question>
      Who was the forty-second
      president of the U.S.A.?
    </question>
    <answer>
      William Jefferson Clinton
    </answer>
  </qanda>
  <!-- Note: We need to add
  more questions later.-->
</quiz>
```

XML

XML

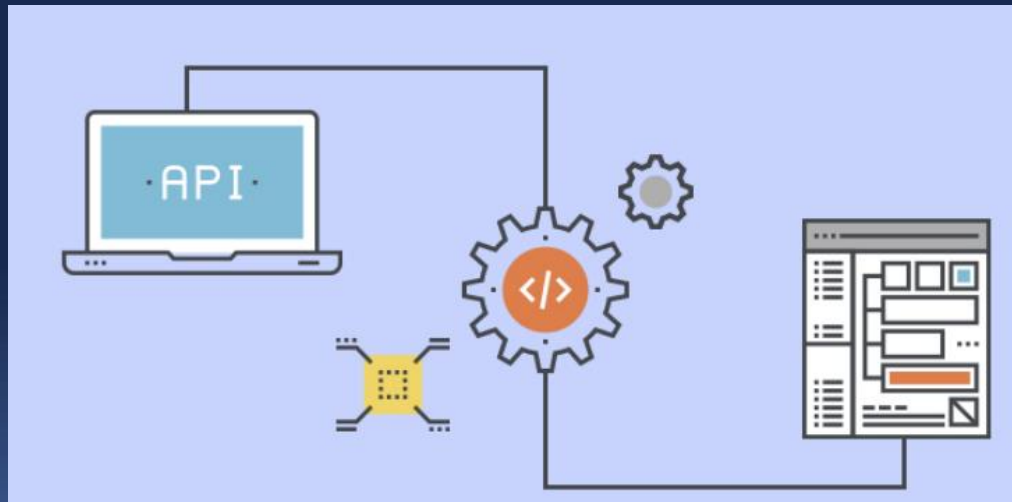
O nome REST

- **R** como vimos está associado à forma de representação dos dados que são interoperacionalizados na integração das aplicações;
- A comunicação que se estabelece é **stateless**;
- Isso significa que como sendo a web baseada no protocolo HTTP, **NÃO** se guarda o estado da aplicação durante a comunicação;
- Ou seja, a comunicação é integralmente **stateless** (sem armazenamento do estado);
- Assim, quando se processa uma **API Rest**, a informação retornada está associada ao estado corrente, sem portanto, se usar seções para armazenamento de dados.



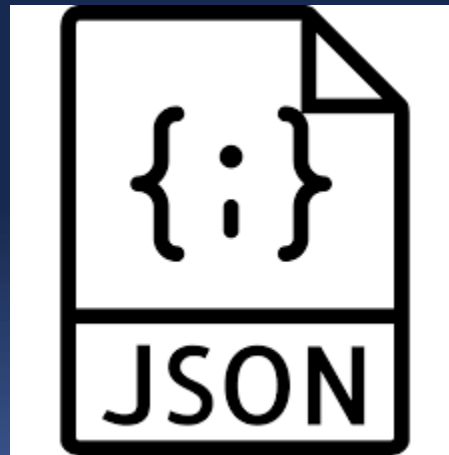
Rest

- Ao contrário do SOAP, **Rest** não precisa usar **XML** para fornecer a resposta;
- Pode-se encontrar REST-based Web Services que entregam os dados em formato CSV (Command Separated Value) ou JSON (JavaScript Object Notation);
- Um ponto **importante** para ser destacado com REST é que se pode obter uma saída em um formato mais fácil de ser tratado na linguagem da aplicação que irá consumir as informações providas pela **API**;



JSON

- É uma formatação **leve** para troca de dados;
- Fácil de **ler** e de se **escrever**;
- Baseia-se em um **subconjunto** da linguagem **JavaScript**;
- Formato texto e completamente **independente** de linguagem;
- Essas propriedades fazem com que JSON seja um formato ideal para troca de dados;



Formato JSON

- Em **JSON** para cada valor representado, atribui-se um nome (ou rótulo) que descreve o seu significado;
- Essa sintaxe é derivada da forma utilizada pelo **JavaScript**, para representar informações;
- Por exemplo, para se representar o ano de 2020, usa-se a seguinte sintaxe:

```
"ano": 2020
```

Formato JSON

- Um par **nome/valor** deve ser representado pelo nome entre aspas duplas, seguido de dois pontos, seguido do valor;
- Os valores podem possuir apenas 3 tipos básicos: **numérico** (inteiro ou real), **booleano** e **string**;

```
"site": "www.qualitsys.com"
```

```
"nota": 9.5
```

```
"aprovado": true
```

Formato JSON

- A partir dos dados básicos, é possível construir-se tipos complexos: array e objeto;
- Arrays são delimitados por **colchetes**, com seus elementos separados por vírgulas;
- Por exemplo, abaixo segue um exemplo de um array representado números:

```
[34,6,888,34,23,124]
```

- Segue outro exemplo de um array representando valores booleanos e outro para strings:

```
[true,true,false,true,false,false]
```

```
["SP","RJ","MG","RS","PR","SC"]
```

Formato JSON

- Segue um exemplo para armazenar uma matriz de inteiros:

```
[  
  [1,8],  
  [5,9],  
  [23,6]  
]
```

Formato JSON

- Objetos são especificados entre **chaves** e podem ser compostos por múltiplos pares nome/valor, por arrays e também por outros objetos;

Desta forma, um objeto JSON pode representar, virtualmente, qualquer tipo de informação.

```
1 {  
2   "titulo": "JSON x XML",  
3   "resumo": "o duelo de dois modelos de representação de informações",  
4   "ano": 2012,  
5   "genero": ["aventura", "ação", "ficção"]  
6 }
```

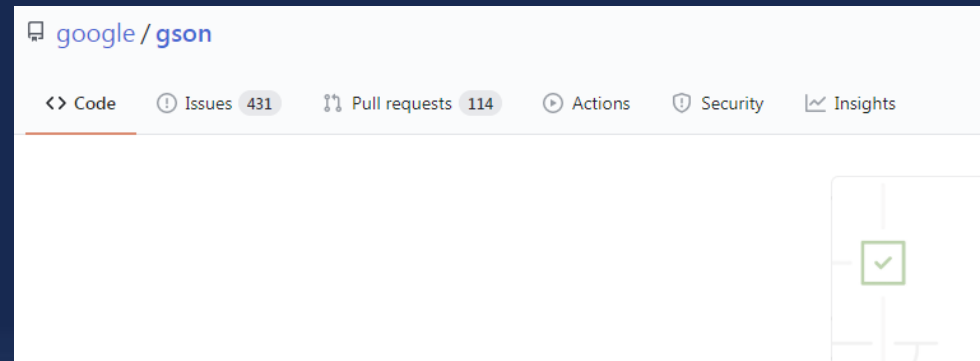
```
1 [  
2   {  
3     "titulo": "JSON x XML",  
4     "resumo": "o duelo de dois modelos de representação de informações",  
5     "ano": 2012,  
6     "genero": ["aventura", "ação", "ficção"]  
7   },  
8   {  
9     "titulo": "JSON James",  
10    "resumo": "a história de uma lenda do velho oeste",  
11    "ano": 2012,  
12    "genero": ["western"]  
13  }  
14 ]
```


Parsers JSON

- Existem diversos **parsers** disponíveis para a Linguagem Java;
- Por exemplo, destaca-se a biblioteca **google-gson**, desenvolvida pela **Google**, que é bem documentada e relativamente simples de ser usada.

Java

- [JSON-java](#)
- JSONUtil
- jsonp
- Json-lib
- Stringtree
- SOJO
- json-taglib
- Flexjson
- Argo
- jsonij
- fastjson
- mjson
- jjson
- json-simple
- json-io
- google-gson**
- FOSS Nova JSON
- Corn CONVERTER
- Apache johnzon
- Genson
- cookjson
- progbase



JSON com Google GSON

- Existem diversos **parsers** disponíveis para a Linguagem Java;
- Por exemplo, destaca-se a biblioteca *gson*, desenvolvida pela **Google**, que é bem documentada e relativamente simples de ser usada.

JSON com Google GSON. **JSON** é um acrônimo para “JavaScript Object Notation”, é um formato mais leve que xml e mais entendível para se trafegar dados entre sistemas computacionais, seu uso está sendo cada vez mais adotado por aplicações web e dispositivos móveis da atualidade.

Algumas das vantagens são:

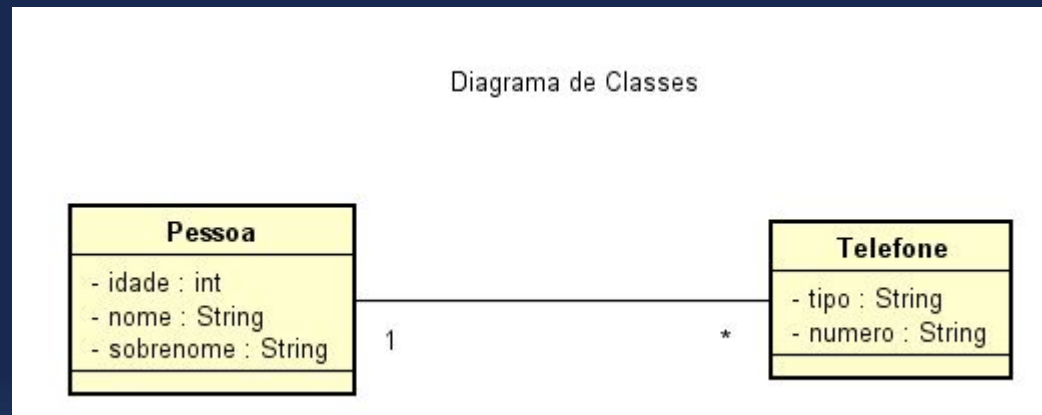
- ✓ Fácil de entender.
- ✓ Parsing facilitado.
- ✓ Suporta objetos.
- ✓ Extremamente leve.
- ✓ Usado pelos maiores serviços da web como Google, Facebook e etc.

JSON é em formato texto e completamente independente de linguagem.

Aplicações web desenvolvidas com Spring MVC e AngularJS fazem o uso extremo de JSON tornando assim uma aplicação leve, robusta, eficiente e com baixo processamento do servidor.

JSON com Google GSON

- Para exemplificar o uso do **google-gson**, vamos ver um caso simples de um relacionamento 1 x n.



JSON com Google GSON

Seguem as implementações das classes:

```
01 public class Pessoa {  
02  
03     private String nome;  
04     private String sobrenome;  
05     private int idade;  
06  
07     private List telefones = new ArrayList();  
08  
09     public void setTelefones(List telefones) {  
10         this.telefones = telefones;  
11     }  
12  
13     public List getTelefones() {  
14         return telefones;  
15     }  
16  
17     public String getNome() {  
18         return nome;  
19     }  
20  
21     public void setNome(String nome) {  
22         this.nome = nome;  
23     }  
24  
25     public String getSobrenome() {  
26         return sobrenome;  
27     }  
28  
29     public void setSobrenome(String sobrenome) {  
30         this.sobrenome = sobrenome;  
31     }  
32  
33     public int getIdade() {  
34         return idade;  
35     }  
36  
37     public void setIdade(int idade) {  
38         this.idade = idade;  
39     }  
}
```

```
public class Telefone {  
  
    private String tipo;  
    private String numero;  
  
    public String getTipo() {  
        return tipo;  
    }  
  
    public void setTipo(String tipo) {  
        this.tipo = tipo;  
    }  
  
    public String getNumero() {  
        return numero;  
    }  
  
    public void setNumero(String numero) {  
        this.numero = numero;  
    }  
}
```

API Gson Google

A referência correta do pacote desta biblioteca é **com.google.gson.Gson** e contém dentre outros com dois métodos que são os mais interessantes e usuais, são eles **toJson** e o **fromJson**. Vamos entender o que cada um faz:

- ✓ **toJson** transforma objetos em JSON com saída String.
- ✓ **fromJson** transforma String JSON em objetos novamente.

Criando os objetos com dados

Nesta parte iremos desenvolver um exemplo simples, criando um objeto pessoa e adicionando telefones a ele.

```
01 Pessoa pessoa = new Pessoa();
02 pessoa.setIdade(29);
03 pessoa.setNome("Java");
04 pessoa.setSobrenome("Avançado");
05
06 Telefone telefone = new Telefone();
07 telefone.setTipo("celular");
08 telefone.setNumero("(44) 5555-8888");
09
10 Telefone telefone2 = new Telefone();
11 telefone2.setTipo("fixo");
12 telefone2.setNumero("(44) 8888-3333");
13
14 pessoa.getTelefones().add(telefone);
15 pessoa.getTelefones().add(telefone2);
```

Convertendo para JSON

Simplesmente instanciamos um objeto Gson e chamamos o método toJson passando a pessoa e o tipo da classe.

```
1 | String json = new Gson().toJson(pessoa, Pessoa.class);
```

O resultado:

```
01 | {  
02 |   "nome": "Java",  
03 |   "sobrenome": "Avançado",  
04 |   "idade": 29,  
05 |   "telefones": [  
06 |     {  
07 |       "tipo": "celular",  
08 |       "numero": "(44) 5555-8888"  
09 |     },  
10 |     {  
11 |       "tipo": "fixo",  
12 |       "numero": "(44) 8888-3333"  
13 |     }  
   ]  
}
```

Convertendo JSON para objeto

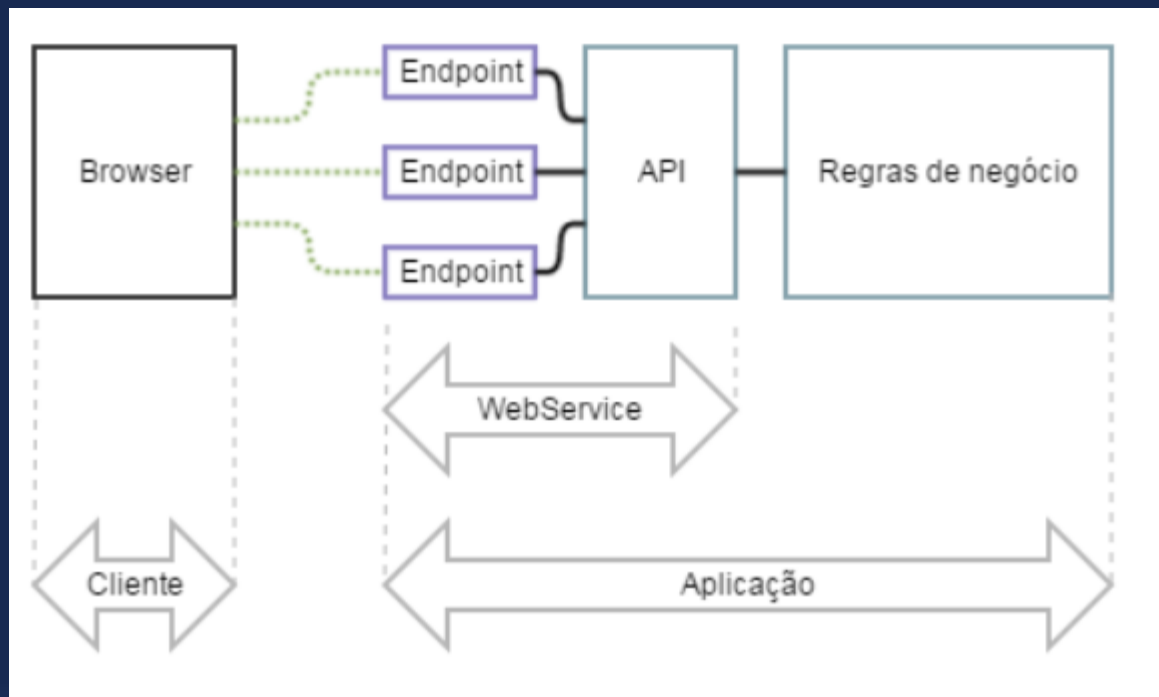
Simplesmente instanciamos um objeto Gson e chamamos o método fromJson passando o JSON e o tipo da classe.

```
01 Pessoa pessoa = new Gson().fromJson("
02 {
03     "nome": "Java",
04     "sobrenome": "Avançado",
05     "idade": 29,
06     "telefones": [
07         {
08             "tipo": "celular",
09             "numero": "(44) 5555-8888"
10         },
11         {
12             "tipo": "fixo",
13             "numero": "(44) 8888-3333"
14         }
15     ], Pessoa.class);
```

endpoint

- De forma simples, um **endpoint** corresponde a um ponto final de um canal de comunicação;
- Quando uma **API** interage com outro sistema, os pontos de contato dessa comunicação são considerados endpoints;
- Para API's um **endpoint** pode incluir uma URL de um servidor ou serviço;
- Cada **endpoint** corresponde ao local para o qual a **API** pode acessar os recursos necessários para processar sua funcionalidade;
- API's operam com requisições e respostas. Quando uma **API** requisita informações de uma aplicação web ou web server, ela receberá uma resposta;
- O local em que as **API's** enviam solicitações e onde o recurso está localizado é chamado **endpoint**.

endpoint





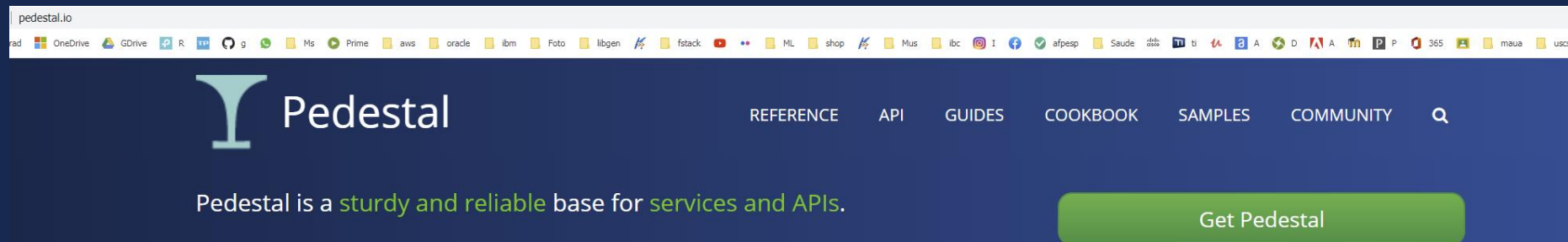
Criação de API's Rest em Clojure com o Framework Pedestal



Setup do Ambiente

- Para a criação da API Rest, utilizaremos a biblioteca Pedestal.

<http://pedestal.io/>



What is Pedestal?

Pedestal is a set of libraries that we use to build services and applications. It runs in the back end and can serve up whole HTML pages or handle API requests.

There are a lot of tools in that space, so why did we build Pedestal? We had two main reasons:

- Pedestal is designed for APIs first. Most web app frameworks still focus on the "page model" and server side rendering. Pedestal lets you start simple and add that if you need it.
- Pedestal makes it easy to create "live" applications. Applications must respond with immediate feedback even while some back-end communication goes on. Pedestal makes it easy to deliver server-sent events and asynchronous updates.

```
(ns front-page
  (:require [io.pedestal.http :as http]))

(defn respond-hello [request]
  {:status 200
   :body "Hello, world!"})

(def routes
  #{["/greet" :get `respond-hello]})

~
~
~
~
~
~
~
~
~
```

O que é Pedestal ?

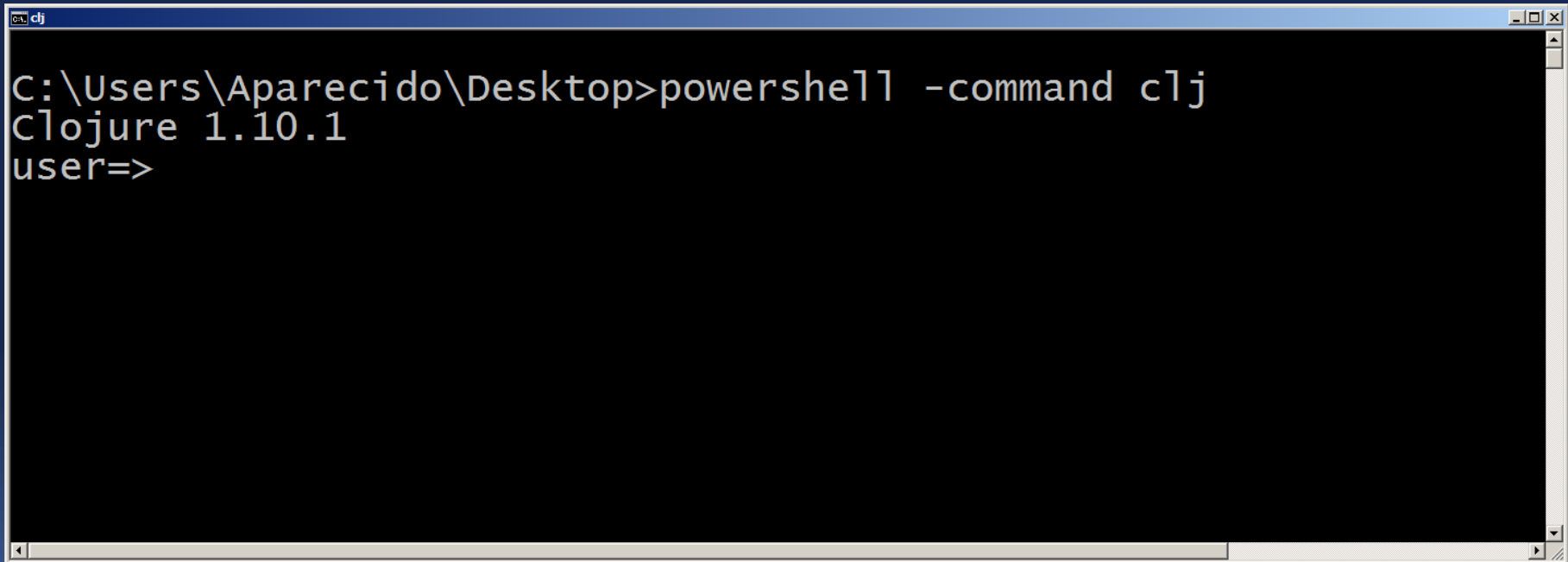
- É um conjunto de bibliotecas que são usadas para se criar serviços e aplicações.
- Roda no **back end** e pode servir requests handle **API**;
- Desenvolveremos nesta unidade uma api simples para demonstrar o uso da biblioteca Pedestal;
- Inicialmente, faremos uma aplicação gerando o projeto a partir do zero !



Criando a pasta do projeto

- Estamos supondo nesta unidade que o clj tools está instalado e operacional;
- Lembrando que na plataforma Windows, clj tools é executado sob Powershell.

➤ `powershell -command clj`

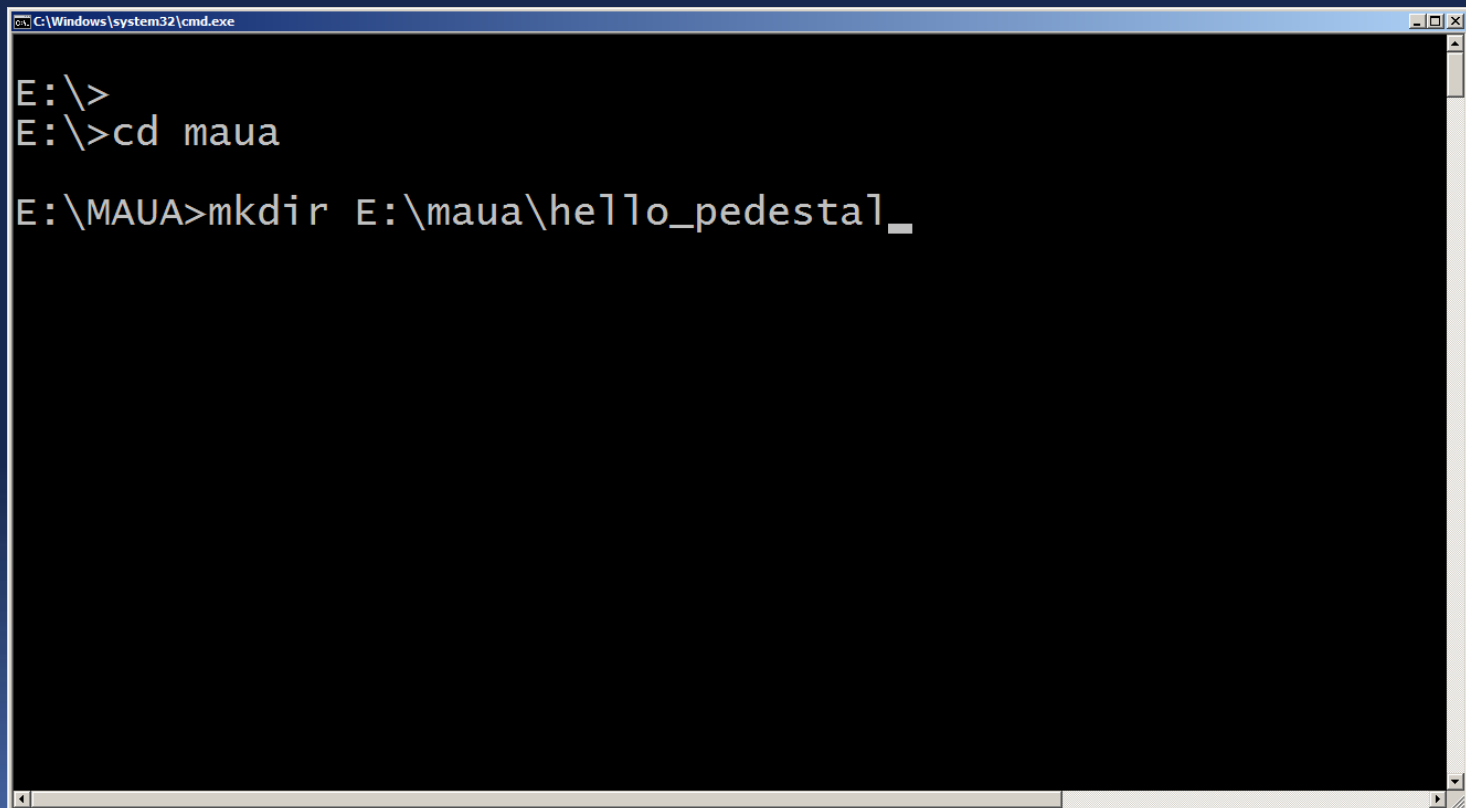


```
cmd
C:\Users\Aparecido\Desktop>powershell -command clj
Clojure 1.10.1
user=>
```

Criando a pasta do Projeto

- Iniciaremos o projeto com um diretório vazio.

- E:
- cd Maua
- mkdir e:\Maua\hello_pedestal

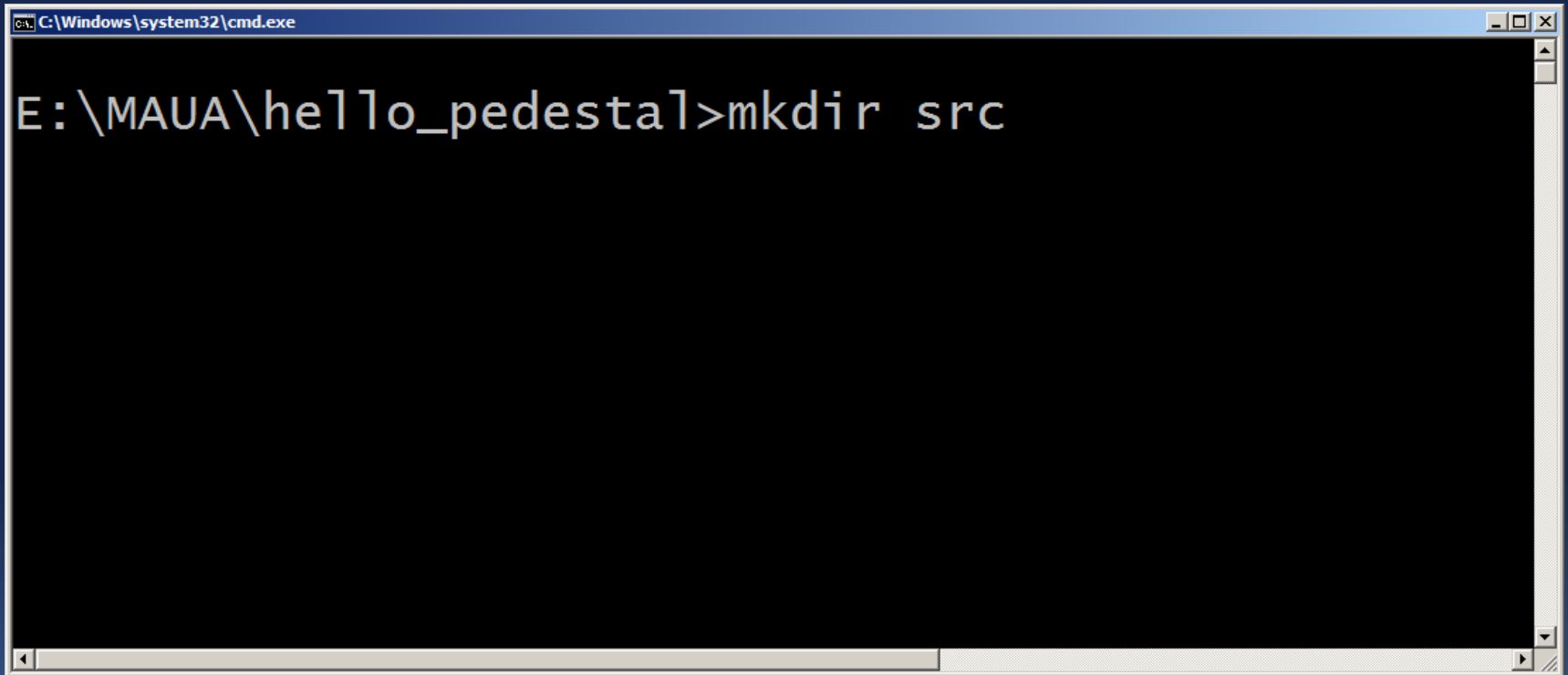


```
C:\Windows\system32\cmd.exe
E:\>
E:\>cd maua
E:\MAUA>mkdir E:\maua\hello_pedestal_
```

Criando a pasta do projeto

- Criaremos um diretório "src" para postarmos o nosso código.

➤ `mkdir src`



```
C:\Windows\system32\cmd.exe  
E:\MAUA\hello_pedestal>mkdir src
```

Escrevendo o código

- No diretório **src**, escreveremos nosso código no arquivo **src/hello-pedestal.clj**;
- Utilizaremos o editor **Atom**, com o plugin **Chlorine**, como visto nas unidades anteriores.

hello-pedestal.clj

```
1 ▾ (ns hello-pedestal)
2 ▾   (:require [io.pedestal.http :as http]
3           [io.pedestal.http.route :as route]))
4
```



Algumas observações do código

hello_pedestal.cj

deps.edn

```
(ns hello_pedestal

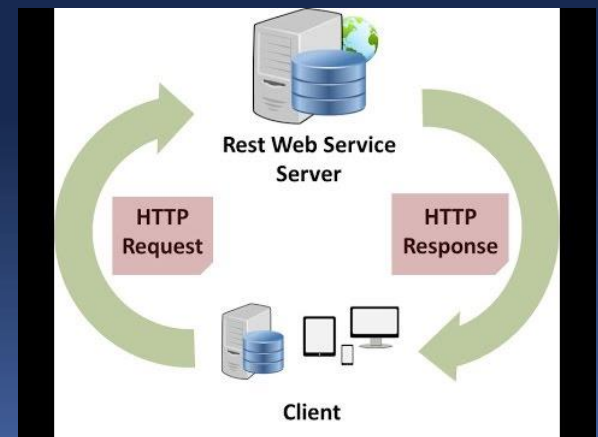
(:require [io.pedestal.http :as http]
           [io.pedestal.http.route :as route]))
```

- ✓ A macro **ns** declara o namespace, semelhantemente a um package em Java. Chamaremos o namespace de "**hello_pedestal**". Quase sempre o nome do namespace coincide com o filename;
- ✓ **:require** é muito similar a um **import** em Java ou **require** em Ruby;
- ✓ O namespace **io.pedestal.http** tem funções que permitem a conexão à servidores HTTP a partir de bibliotecas Pedestal e **io.pedestal.http.route** para roteamentos;
- ✓ **:as** define **alias** de forma que **io.pedestal.http** será escrito simplesmente por **http**.



Gerando uma resposta

- ✓ Construiremos um web service que responda **"Hello Pedestal . . . "** e para isso necessitamos fazer algumas coisas básicas:
 - ❖ Ouvir em um socket **requisições HTTP**;
 - ❖ Compreender o que um dado request **significa**;
 - ❖ Providenciar uma **resposta** para o request.



Gerando uma resposta

- ✓ Faremos todas essas coisas, mas ao contrário;
- ✓ Em **Clojure**, geralmente encontramos as funções mais importantes (high level) na parte inferior do código;
- ✓ Assim, geralmente, quando se lê código **Clojure** pode-se começar a leitura na parte inferior do código;
- ✓ Portanto, o que faremos agora é escrever a função que gera a **response** "Hello Pedestal ..."



Codificando hello-pedestal.clj



```
hello_pedestal.clj      deps.edn

(ns hello_pedestal

  (:require [io.pedestal.http :as http]
             [io.pedestal.http.route :as route]))

(defn resposta [request]
  {:status 200 :body "Hello, Pedestal..."})
```

- ✓ Definiremos uma função chamada **resposta** que recebe um argumento, o qual estamos chamando de **request**;
- ✓ A função retorna um **map** com duas chaves e dois valores (**:status 200** e **:body "Hello, Pedestal !!!"**)

Gerenciando dependências



- ✓ **Pedestal** é um grande projeto de tecnologia open-source e, por conta disso, possui diversos módulos os quais precisamos gerenciar as dependências;
- ✓ Para trabalharmos com essas bibliotecas de dependências, precisamos declará-las como dependências, de forma que possam ser baixadas e adicionadas ao class-path. Criaremos para isso, um arquivo chamado **deps.edn** na pasta do projeto.

deps.edn

hello-pedestal.clj

```
1
2  { :deps
3    {io.pedestal/pedestal.service { :mvn/version "0.5.7"}
4    io.pedestal/pedestal.route    { :mvn/version "0.5.7"}
5    io.pedestal/pedestal.jetty    { :mvn/version "0.5.7"}
6    org.slf4j/slf4j-simple        { :mvn/version "1.7.28"}}
7    :paths ["src"]}
```

Gerenciando dependências



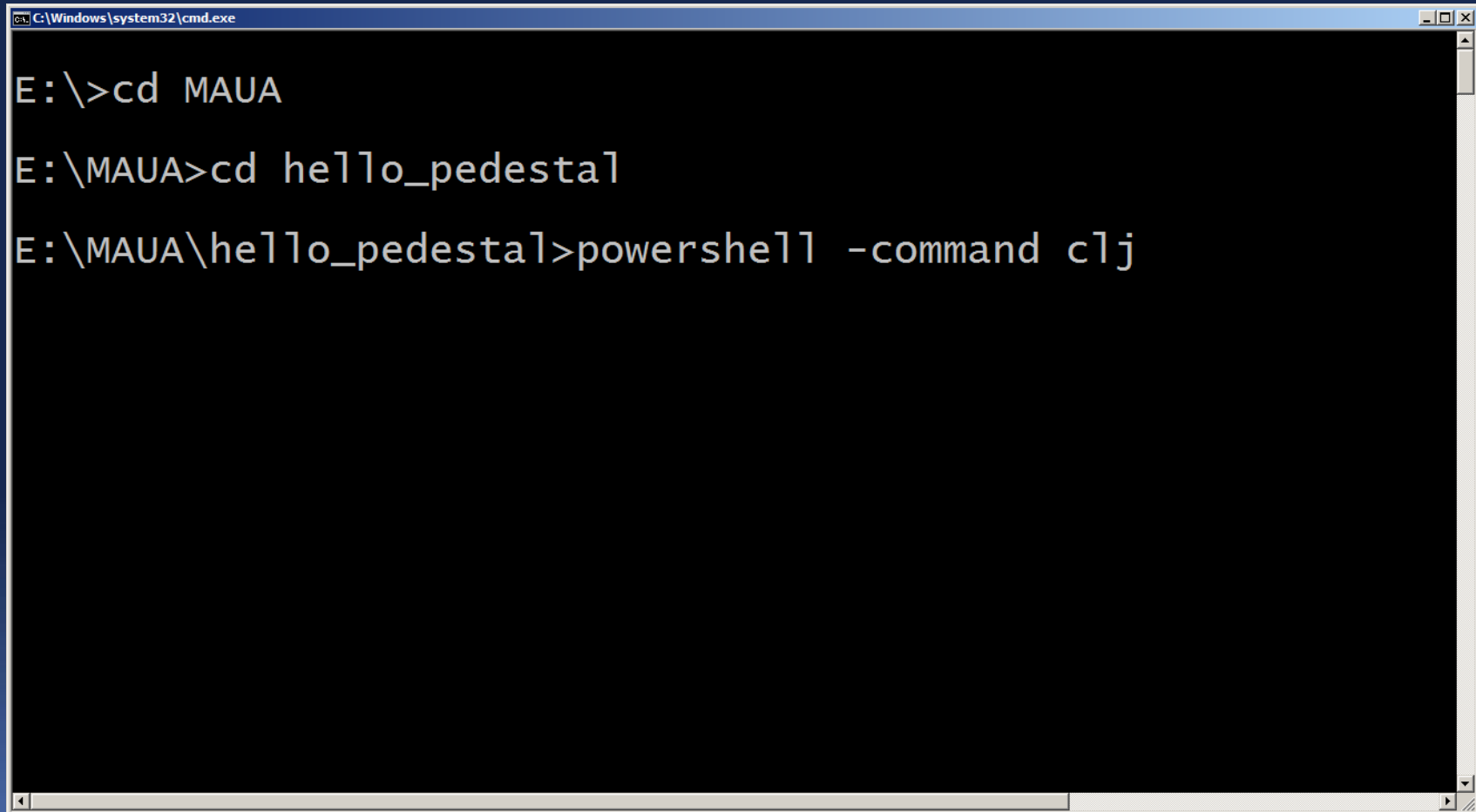
```
deps.edn      hello-pedestal.clj
1
2  { :deps
3    { io.pedestal/pedestal.service { :mvn/version "0.5.7" }
4      io.pedestal/pedestal.route   { :mvn/version "0.5.7" }
5      io.pedestal/pedestal.jetty   { :mvn/version "0.5.7" }
6      org.slf4j/slf4j-simple       { :mvn/version "1.7.28" } }
7    :paths ["src"] }
```

- ✓ Estaremos baixando três bibliotecas: as duas primeiras já foram explicadas anteriormente;
- ✓ Pedestal trabalha com diversos servidores **HTTP**. Usaremos nesse exemplo, o servidor Jetty que é um servidor **HTTP** rápido e estável;
- ✓ Além disso, **Jetty** não requer qualquer instalação, podemos startá-lo internamente ao nosso serviço, fazendo com que nosso serviço seja auto-contido e portátil.

Executando a função resposta



- ✓ Executaremos o nosso código com **clj tools**, assumindo que está instalado e operacional;
- ✓ Na pasta do projeto executaremos **clj tools** sob **Powershell**.

A screenshot of a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. The window has a black background with white text. The command history shows the user navigating to the 'MAUA' directory, then to the 'hello_pedestal' subdirectory, and finally running 'powershell -command clj'.

```
C:\Windows\system32\cmd.exe

E:\>cd MAUA

E:\MAUA>cd hello_pedestal

E:\MAUA\hello_pedestal>powershell -command clj
```

Executando a função resposta



Pedestal

- ✓ Executaremos o nosso código com **clj tools**, assumindo que está instalado e operacional;
- ✓ Na pasta do projeto executaremos **clj tools** sob **Powershell**;
- ✓ **clj tools** baixará as dependências do projeto, as adicionará ao classpath e abrirá um REPL no namespace **user**.

```
C:\Windows\system32\cmd.exe - powershell -command clj

E:\MAUA\hello_pedestal>powershell -command clj
Clojure 1.10.1
user=> _
```



Executando a função resposta

- ✓ Agora estamos prontos para processar o código;
- ✓ A primeira coisa a fazer é importar o namespace **hello_pedestal**

➤ `(require 'hello_pedestal)`

```
C:\Windows\system32\cmd.exe - powershell -command clj
E:\>cd MAUA
E:\MAUA>cd hello_pedestal
E:\MAUA\hello_pedestal>powershell -command clj
Clojure 1.10.1
user=> (require 'hello_pedestal)
nil
user=>
```


Executando a função resposta



✓ Executando a função resposta

➤ (hello_pedestal/resposta)

```
C:\Windows\system32\cmd.exe - powershell -command clj
user=>
user=>
user=>
user=> (require 'hello_pedestal)
nil
user=>
user=>
user=>
user=>
user=>
user=> (hello_pedestal/resposta {})
{:status 200, :body "Hello, Pedestal..."}
user=>
user=>
user=>
```

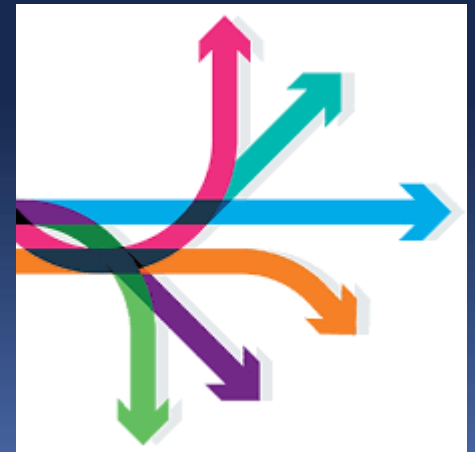




Pedestal

Conectando o código a uma Rota

- ✓ No Pedestal, o roteamento é o processo de mapear um request recebido a um **handler** (manuseador);
- ✓ Em nosso exemplo, podemos usar o nosso código como um **handler**;
- ✓ Diremos ao Pedestal que a rota **"/ola"** mapeará nossa função handler.



Conectando o código a uma Rota



Pedestal

- ✓ O roteamento está associando requests **HTTP GET** com a query string **"/ola"** à função resposta. O nome do roteamento é **:ola**;

```
(def routes
  (route/expand-routes
    #{["/ola" :get resposta :route-name :ola]}))
```



Definindo um servidor

```
(defn create-server []  
  (http/create-server  
    { ::http/routes routes  
      ::http/type :jetty  
      ::http/port 8899}))
```



Função para iniciar o servidor

```
(defn start []  
  (http/start (create-server)))
```



Recarregando o namespace

```
C:\Windows\system32\cmd.exe - powershell -command cj
user=>
user=>
user=>
user=>
user=> (require :reload 'hello_pedestal)
nil
user=>
```



Iniciando o Servidor

➤ (hello_pedestal/start)

```
C:\Windows\system32\cmd.exe - powershell -command clj
user=>
user=>
user=>
user=> (hello_pedestal/start)
[main] INFO org.eclipse.jetty.util.log - Logging initialized @
clipse.jetty.util.log.Slf4jLog
[main] INFO org.eclipse.jetty.server.Server - jetty-9.4.18.v20
9-04-29T20:42:08.989Z; git: e1bc35120a6617ee3df052294e433f3a25
241-b07
[main] INFO org.eclipse.jetty.server.handler.ContextHandler -
rvletContextHandler@7a0ebd12{/,null,AVAILABLE}
[main] INFO org.eclipse.jetty.server.AbstractConnector - Start
@70347322{HTTP/1.1,[http/1.1, h2c]}{localhost:8899}
[main] INFO org.eclipse.jetty.server.Server - Started @5425028
```



Servidor rodando...

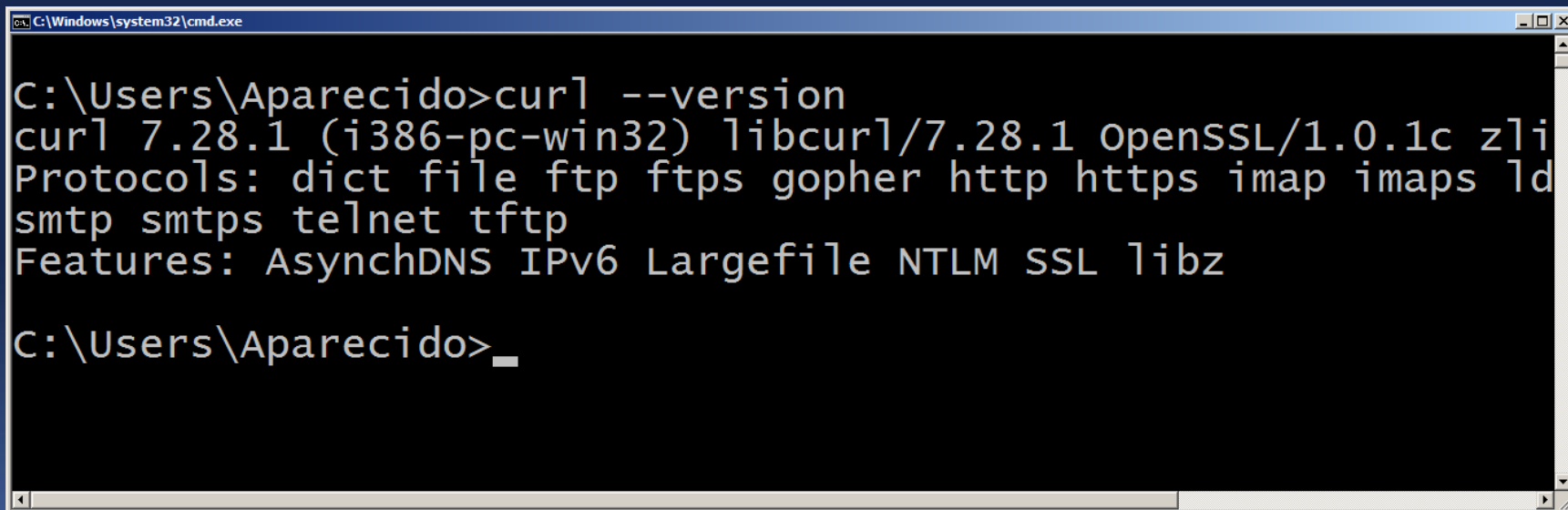
```
C:\Windows\system32\cmd.exe - powershell -command cd
user=>
user=>
user=>
user=> (hello_pedestal/start)
[main] INFO org.eclipse.jetty.util.log - Logging initialized @
clipse.jetty.util.log.Slf4jLog
[main] INFO org.eclipse.jetty.server.Server - jetty-9.4.18.v20
9-04-29T20:42:08.989Z; git: e1bc35120a6617ee3df052294e433f3a25
241-b07
[main] INFO org.eclipse.jetty.server.handler.ContextHandler -
rvletContextHandler@7a0ebd12{/,null,AVAILABLE}
[main] INFO org.eclipse.jetty.server.AbstractConnector - Start
@70347322{HTTP/1.1,[http/1.1, h2c]}{localhost:8899}
[main] INFO org.eclipse.jetty.server.Server - Started @5425028
```

- ✓ O servidor **HTTP Jetty** está rodando e aguardando **requests**...



Criando um request com curl

- ✓ **Curl** é um comando disponível na maioria dos sistemas UNIX;
- ✓ **Curl** é uma abreviação de **Client URL**;
- ✓ Por meio do comando **Curl** podemos transferir dados para um servidor **HTTP** através de um request (cliente).



```
C:\Windows\system32\cmd.exe

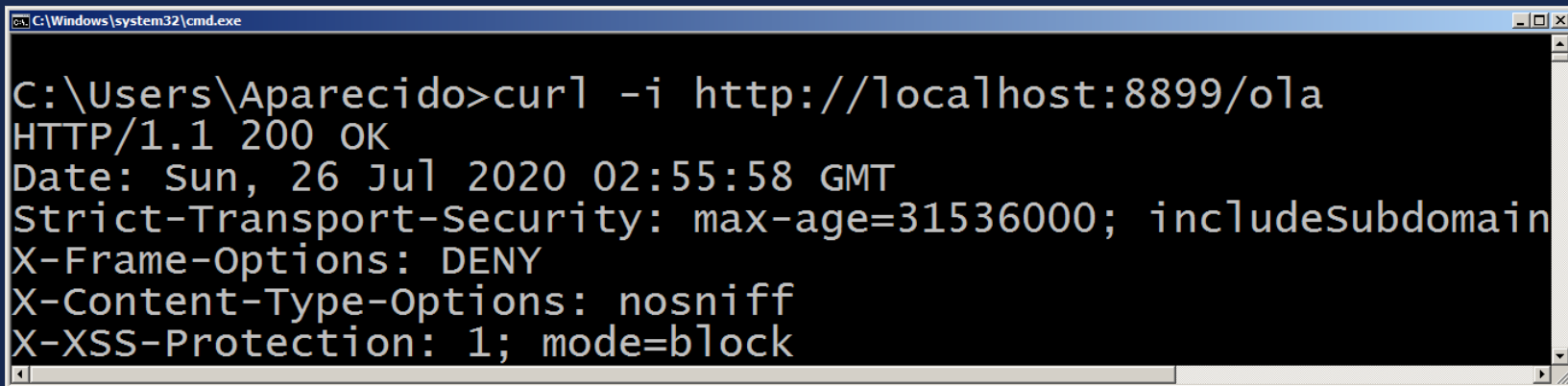
C:\Users\Aparecido>curl --version
curl 7.28.1 (i386-pc-win32) libcurl/7.28.1 OpenSSL/1.0.1c zlib
Protocols: dict file ftp ftps gopher http https imap imaps ldap
smtp smtps telnet tftp
Features: AsynchDNS IPv6 Largefile NTLM SSL libz

C:\Users\Aparecido>_
```



Criando um request com curl

➤ `curl -i http://localhost:8899/ola`



```
C:\Windows\system32\cmd.exe

C:\Users\Aparecido>curl -i http://localhost:8899/ola
HTTP/1.1 200 OK
Date: Sun, 26 Jul 2020 02:55:58 GMT
Strict-Transport-Security: max-age=31536000; includeSubdomain
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
```



Criando um request com curl

➤ `curl -i http://localhost:8899/ola`



```
C:\Windows\system32\cmd.exe
Content-Type: text/plain
Transfer-Encoding: chunked

Hello, Pedestal...
C:\Users\Aparecido>
```





Pedestal

Recriando o projeto com Clojure, Pedestal, Leiningen, Jetty e Atom - Chlorine



Pedestal

**jetty://**



Assume-se que Leiningen está instalado



➤ lein

```
Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Aparecido>lein
Leiningen is a tool for working with Clojure projects.

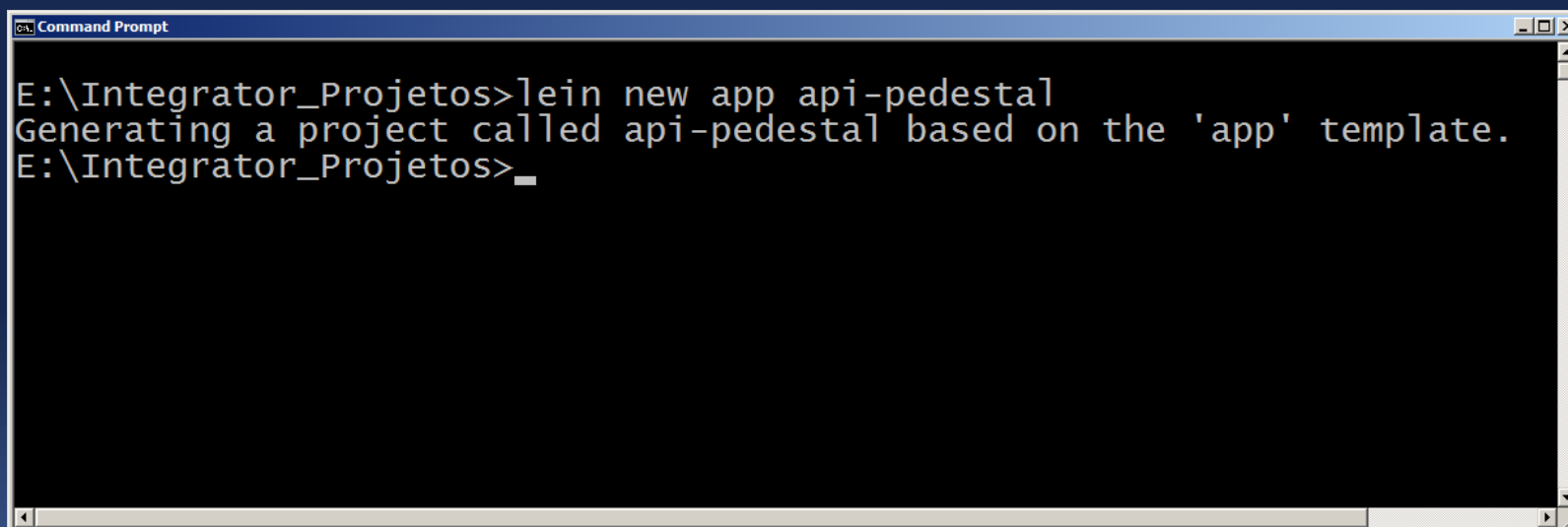
Several tasks are available:
change      Rewrite project.clj by applying a function.
check       Check syntax and warn on reflection.
classpath   Print the classpath of the current project.
clean       Remove all files from project's target-path.
compile     Compile Clojure source into .class files.
deploy      Build and deploy jar to remote repository.
```





Criando uma nova aplicação com Leiningen

- ✓ A partir do Leiningen criaremos uma nova aplicação, por meio do template **app**, e a chamaremos **api-pedestal** no diretório E:\Integrator_Projetos.
 - E:
 - `cd\integrator_projetos`
 - `lein new app api-pedestal`



```
Command Prompt
E:\Integrator_Projetos>lein new app api-pedestal
Generating a project called api-pedestal based on the 'app' template.
E:\Integrator_Projetos>
```



Criando uma nova aplicação com Leiningen

- ✓ A pasta do projeto foi criada pelo **Leiningen**;
- ✓ Essa pasta já está estruturada com um **padrão** de projeto;
- ✓ Um arquivo especial de configuração do projeto chamado **project.clj** também foi criado no diretório.

```
Command Prompt

Directory of E:\Integrator_Projetos\api-pedestal

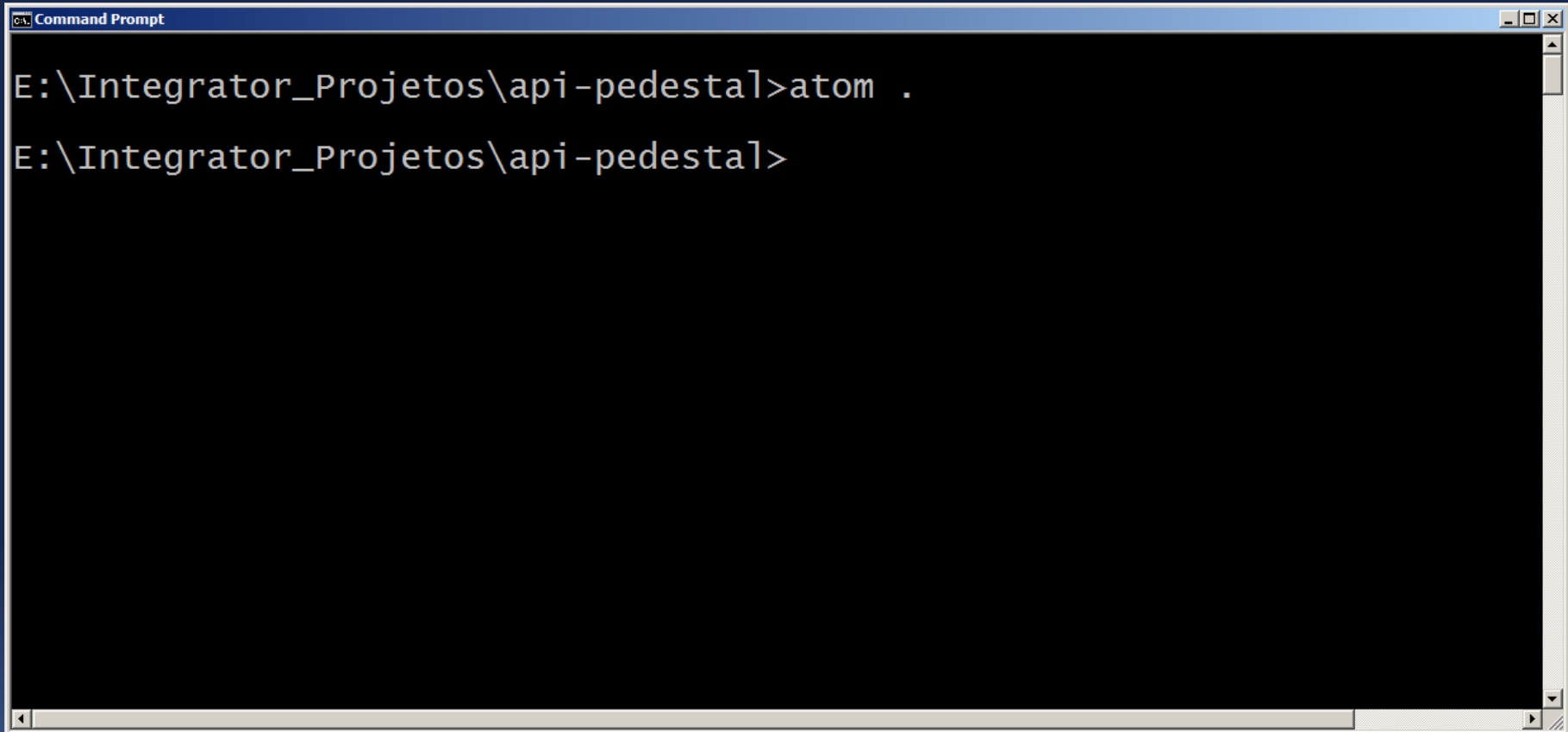
26-Jul-20  10:07 AM    <DIR>          .
26-Jul-20  10:07 AM    <DIR>          ..
26-Jul-20  10:07 AM             124 .gitignore
26-Jul-20  10:07 AM             164 .hgignore
26-Jul-20  10:07 AM             802 CHANGELOG.md
26-Jul-20  10:07 AM    <DIR>          doc
26-Jul-20  10:07 AM      14,652 LICENSE
26-Jul-20  10:07 AM       412 project.clj
26-Jul-20  10:07 AM      1,031 README.md
26-Jul-20  10:07 AM    <DIR>          resources
26-Jul-20  10:07 AM    <DIR>          src
26-Jul-20  10:07 AM    <DIR>          test
                6 File(s)             17,185 bytes
                6 Dir(s)  350,336,331,776 bytes free

E:\Integrator_Projetos\api-pedestal>
```



Subindo o Atom a partir do diretório

➤ `atom .`

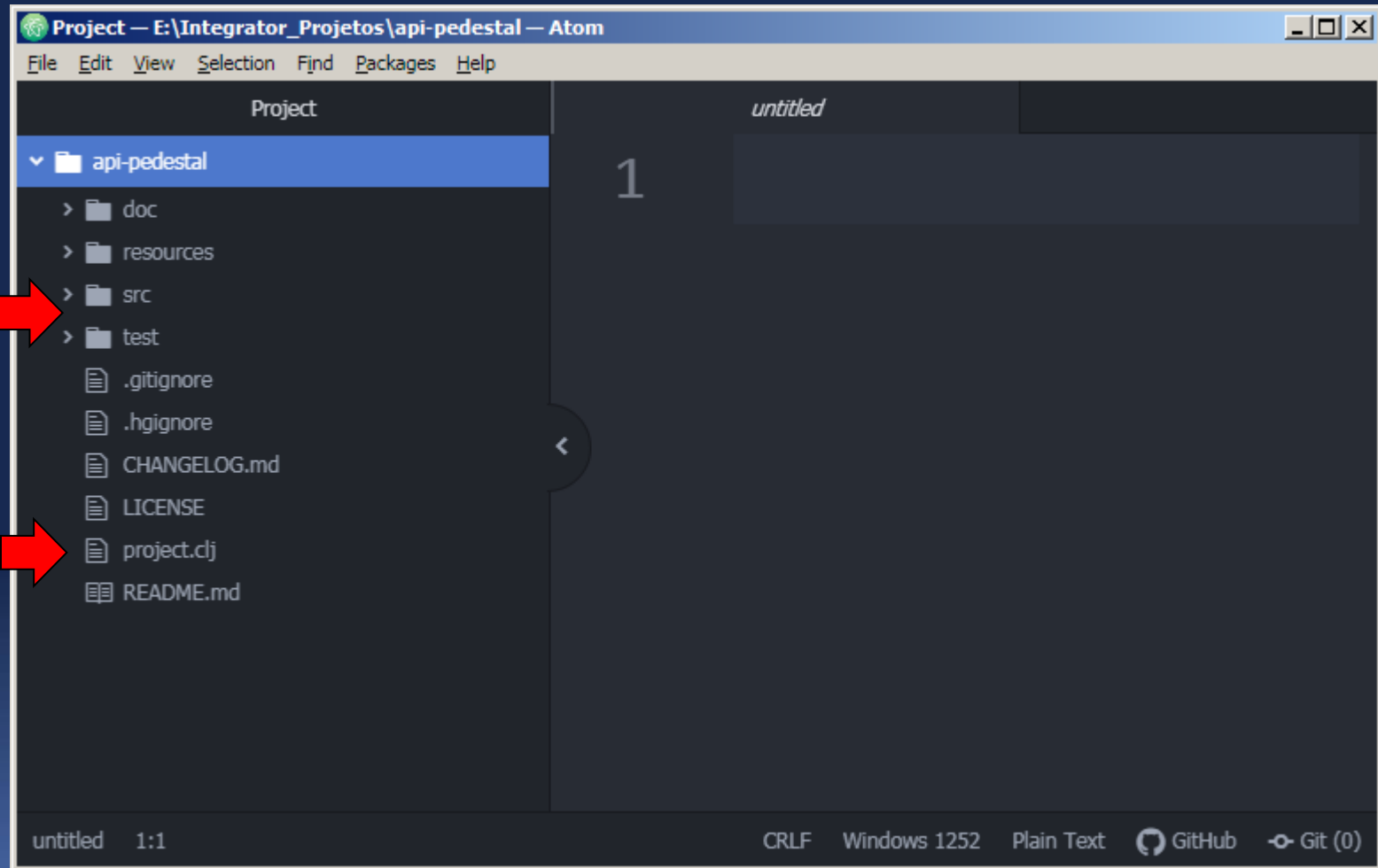


```
Command Prompt
E:\Integrator_Projetos\api-pedestal>atom .
E:\Integrator_Projetos\api-pedestal>
```




Subindo o Atom a partir do diretório

- ✓ Pode-se visualizar toda a estrutura do nosso projeto !





Configurações do Projeto

```
Project -- E:\Integrator_Projetos\api-pedestal -- Atom
File Edit View Selection Find Packages Help

Project
  api-pedestal
  doc
  resources
  src
  test
  .gitignore
  .hgignore
  CHANGELOG.md
  LICENSE
  project.clj
  README.md

1 (defproject api-pedestal "0.1.0-SNAPSHOT"
2   :description "FIXME: write description"
3   :url "http://example.com/FIXME"
4   :license {:name "EPL-2.0 OR GPL-2.0-or-later WITH Classpath-exception-2.0"
5           :url "https://www.eclipse.org/legal/epl-2.0/"}
6   :dependencies [[org.clojure/clojure "1.10.1"]]
7   :main ^:skip-aot api-pedestal.core
8   :target-path "target/%s"
9   :profiles {:uberjar {:aot :all}})
10
```

✓ Versão 1.10.1 do Clojure;

Incluindo as dependências do Pedestal



Pedestal

- ✓ Estaremos seguindo nesse projeto as recomendações definidas em <http://pedestal.io/guides/hello-world>
- ✓ As configurações de dependência do Pedestal retiradas da página do Pedestal, serão incluídas no arquivo **project.clj**

```
1 v (defproject api-pedestal "0.1.0-SNAPSHOT"
2   :description "FIXME: write description"
3   :url "http://example.com/FIXME"
4   :license {:name "EPL-2.0 OR GPL-2.0-or-later WITH Classpath-exception-2.0"
5             :url "https://www.eclipse.org/legal/epl-2.0/" }
6   :dependencies [ [org.clojure/clojure "1.10.1"]
7                  [io.pedestal/pedestal.service "0.5.7"]
8                  [io.pedestal/pedestal.route "0.5.7"]
9                  [io.pedestal/pedestal.jetty "0.5.7"] ]
10  :main ^:skip-aot api-pedestal.core
11  :target-path "target/%s"
12  :profiles {:uberjar {:aot :all}})
```





Acrescentando um profile **dev** ao projeto

- ✓ Estaremos incluindo no projeto algumas dependências específicas para a área de Desenvolvimento.

```
project.clj      core.clj

(defproject api-pedestal "0.1.0-SNAPSHOT"
  :description "FIXME: write description"
  :url "http://example.com/FIXME"
  :license {:name "EPL-2.0 OR GPL-2.0-or-later WITH Classpath-exception-2.0"
            :url "https://www.eclipse.org/legal/epl-2.0/"}
  :dependencies [ [org.clojure/clojure "1.10.1"]
                  [io.pedestal/pedestal.service "0.5.7"]
                  [io.pedestal/pedestal.route "0.5.7"]
                  [io.pedestal/pedestal.jetty "0.5.7"] ]
  :main ^:skip-aot api-pedestal.core
  :target-path "target/%s"
  :profiles { :uberjar {:aot :all}
              :dev {:dependencies []
                    :source-paths ["dev"]}}})
```





Acrescentando a pasta "dev" no projeto

▼ api-pedestal

> dev

> doc

> resources

> src

> test

📄 .gitignore

📄 .hgignore

📄 CHANGELOG.md

📄 LICENSE

📄 project.clj

📄 README.md



Acrescentando configuração para Chlorine

- ✓ Estaremos incluindo no projeto a profile :jvm-opts para que operarmos com **REPL** remoto no **Atom**, via plugin **Chlorine**.

```
project.clj

(defproject api-pedestal "0.1.0-SNAPSHOT"
  :description "FIXME: write description"
  :url "http://example.com/FIXME"
  :license {:name "EPL-2.0 OR GPL-2.0-or-later WITH Classpath-exception-2.0"
            :url "https://www.eclipse.org/legal/epl-2.0/"}
  :repl-options {:init-ns user}
  :dependencies [ [org.clojure/clojure "1.10.1"]
                  [io.pedestal/pedestal.service "0.5.7"]
                  [io.pedestal/pedestal.route "0.5.7"]
                  [io.pedestal/pedestal.jetty "0.5.7"]
                  [org.slf4j/slf4j-simple "1.7.28"] ]
  :jvm-opts ["-Dclojure.server.myrepl={:port,5555,:accept,clojure.core.server/repl}"]
  :main ^:skip-aot api-pedestal.core
  :target-path "target/%s"
  :profiles { :uberjar {:aot :all
                        :jvm-opts ["-Dclojure.compiler.direct-linking=true"]}
              :dev {:dependencies []
                    :source-paths ["dev"]}}})
```



Subindo o repl

- ✓ Na pasta do projeto executar: **> lein repl**

```
Command Prompt - lein repl

E:\Integrator_Projetos\api-pedestal>lein repl
nREPL server started on port 61883 on host 127.0.0.1 - nrepl://127.0.0.1:61883
REPL-y 0.4.4, nREPL 0.6.0
Clojure 1.10.1
Java HotSpot(TM) 64-Bit Server VM 1.8.0_241-b07
  Docs: (doc function-name-here)
        (find-doc "part-of-name-here")
  Source: (source function-name-here)
  Javadoc: (javadoc java-object-or-class-here)
  Exit: Control+D or (exit) or (quit)
  Results: Stored in vars *1, *2, *3, an exception in *e

user=> ~
```



Conectando o REPL remoto no Atom

- ✓ No Atom, teclar: **Ctrl + Shift + P** => Chroline: **Connect Socket Repl**





Conectando o REPL remoto no Atom

- ✓ Digitar a porta 5555, conforme definido no projeto (project.clj)

Connect to Socket REPL

Host:

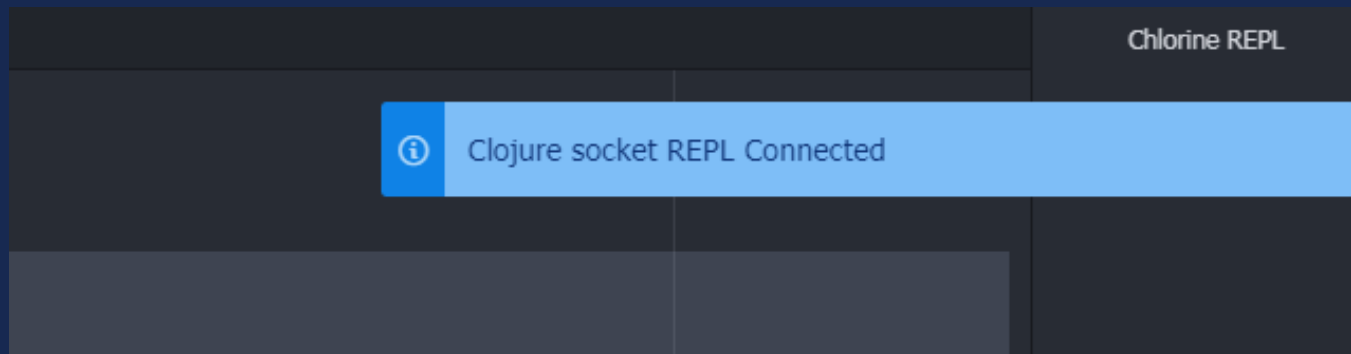
localhost

Port:

5555



REPL remoto conectado !





Console REPL liberada no Atom !



The screenshot shows the Atom editor interface with the file `core.cj` open. The left sidebar displays the project structure for `api-pedestal`, including folders like `dev`, `doc`, `resources`, `src`, `target`, and `test`, as well as files like `.gitignore`, `.hgignore`, `.lein-repl-history`, `.nrepl-port`, `CHANGELOG.md`, `LICENSE`, `project.cj`, and `README.md`. The main editor area shows the code in `core.cj`:

```
1 (ns api-pedestal.core)
2   (:gen-class)
3
4 (defn -main
5   "I don't do a whole lot"
6   [& args]
7   (println "Hello, World!"))
8
```

On the right side, the `Chlorine REPL` console is visible. A red arrow points to the REPL console, indicating that the REPL is now available within the Atom editor.

Acessando o ponto de entrada da Aplicação

✓ Pasta src: **core.clj**

```
Chlorine REPL — E:\Integrator_Projetos\api-pedestal — Atom
File Edit View Selection Find Packages Help

Project
  api-pedestal
    dev
    doc
    resources
    src
      api_pedestal
        core.clj
    target
    test
    .gitignore
    .hgignore
    .lein-repl-history
    .nrepl-port
    CHANGELOG.md
    LICENSE
    project.clj
    README.md

core.clj
1 (ns api-pedestal.core
2   (:gen-class))
3
4 (defn -main
5   "I don't do a whole lot ... yet."
6   [& args]
7   (println "Hello, World!"))
8

Chlorine REPL

Chlorine REPL CLJ (simple) GitHub Git (0)
```

Adicionando **requires** do Pedestal

- ✓ Como vimos, **require** em **Clojure** corresponde aos **imports** do Java, para acesso à bibliotecas externas;

```
core.clj
(ns api-pedestal.core
  (:gen-class)
  (:require [io.pedestal.http :as http]
             [io.pedestal.http.route :as route]))

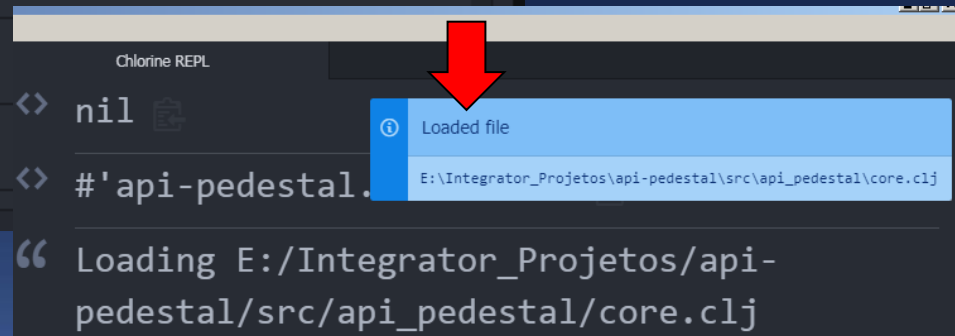
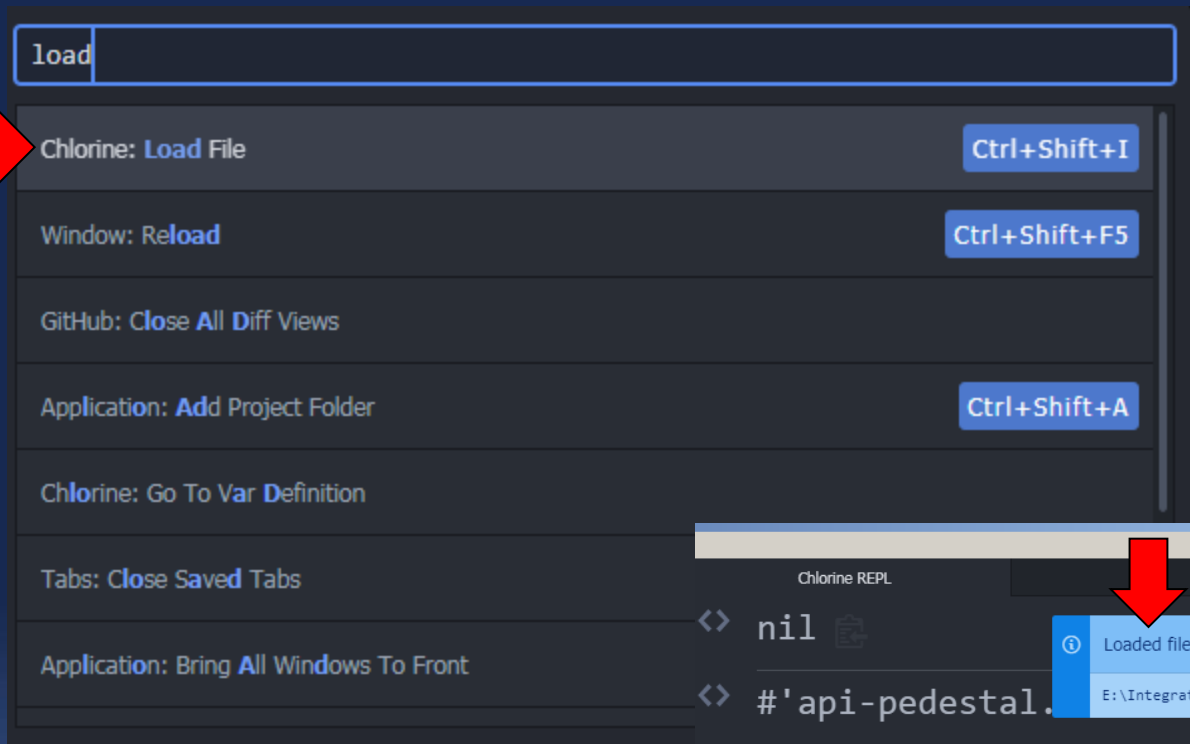
nil

(defn -main
  "I don't do a whole lot ... yet."
  [& args]
  (println "Hello, World!"))

#'api-pedestal.core/-main ...
```

Carregando fonte Clojure no Atom

- ✓ Para isso usaremos o comando Load-File do Plugin Chroline;
- ✓ No editor Atom => **Ctrl+Shift+P > load-file**



Carregando fonte Clojure no Atom

- ✓ Aproveitaremos o “**Hello World**” apresentado no guides do Pedestal ;
- ✓ Detalhes deste exemplo estão em: <http://pedestal.io/guides/hello-world>

core.clj	Chlorine REPL
<pre>(ns api-pedestal.core (:gen-class) (:require [io.pedestal.http :as http] [io.pedestal.http.route :as route])) nil (defn -main "I don't do a whole lot ... yet." [& args] (println "Hello, World!")) #'api-pedestal.core/-main ...</pre>	<pre><> nil <> #'api-pedestal.core/-main ... “ Loading E:/Integrator_Projetos/api-pedestal/src/api_pedestal/core.clj Loading E:/Integrator_Projetos/api-pedestal/src/api_pedestal/core.clj</pre>

Codificando a resposta

```
core.clj                                     Chlorine REPL

▼ (ns api-pedestal.core
  (:gen-class)
  ▼ (:require [io.pedestal.http :as http]
      [io.pedestal.http.route :as route]))

  ▼ (defn respond-hello [request]
      {:status 200   :body "Hello, Pedestal ..."})

  ▼ (defn -main
      "I don't do a whole lot ... yet."
      [& args]
      (println "Hello, World!"))
```

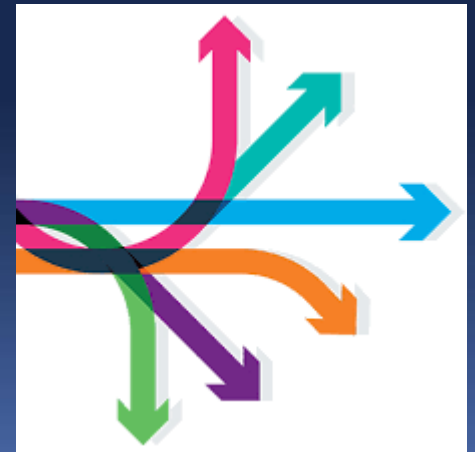
- ✓ Definiremos uma função chamada **respond-hello** que recebe um argumento, o qual estamos chamando de **request**;
- ✓ A função retorna um **map** com duas chaves e dois valores (**:status 200** e **:body "Hello, Pedestal !!!"**)



Pedestal

Conectando o código a uma Rota

- ✓ No Pedestal, o roteamento é o processo de mapear um request recebido a um **handler** (manuseador);
- ✓ Em nosso exemplo, podemos usar o nosso código como um **handler**;
- ✓ Diremos ao Pedestal que a rota **"/greet"** mapeará nossa função handler.



Conectando o código a uma Rota



Pedestal

- ✓ O roteamento está associando requests HTTP GET com a query string **"/greet"** à função resposta. O nome do roteamento é **:greet**;

```
core.clj

(ns api-pedestal.core
  (:gen-class)
  (:require [io.pedestal.http :as http]
             [io.pedestal.http.route :as route]))

(defn respond-hello [request]
  {:status 200 :body "Hello, Pedestal ..."})

(def routes
  (route/expand-routes
   #{["/greet" :get respond-hello :route-name :greet]}))

(defn -main
  "I don't do a whole lot ... yet."
  [& args]
  (println "Hello, World!"))
```

Definindo um servidor



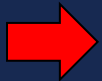
Pedestal

```
(defn server []  
  (http/create-server  
    { ::http/routes routes  
      ::http/type :jetty  
      ::http/port 3002}))  
  
(defn start []  
  (http/start (server)))  
  
(defn stop []  
  (http/stop (server)))  
  
(start)  
  
(stop)
```

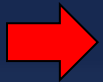


Função para iniciar o servidor

```
(defn start []  
  (http/start (create-server)))
```



```
(start)
```



✓ No Chroline, teclar: Ctrl + Enter, para processar (start)



Servidor rodando na porta 3002

```
Command Prompt - lein repl

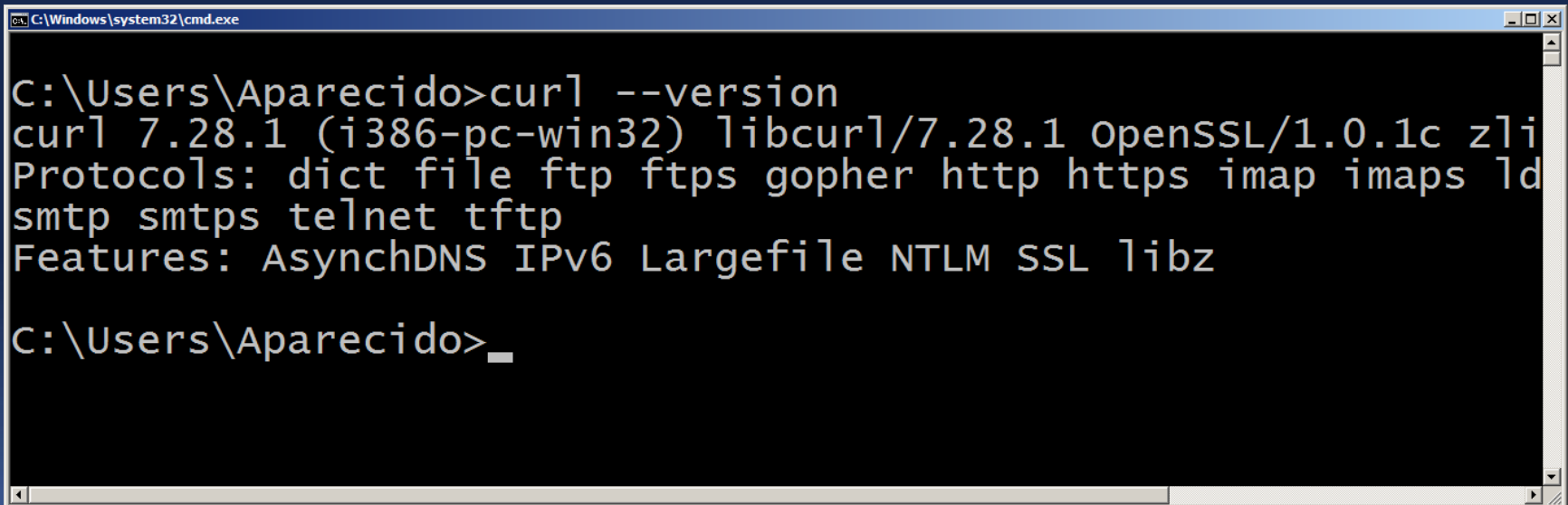
user=> [clojure-agent-send-off-pool-3] INFO org.eclipse.jetty.util.log - Logging
        initialized @65944ms to org.eclipse.jetty.util.log.Slf4jLog
[clojure-agent-send-off-pool-3] INFO org.eclipse.jetty.server.Server - jetty-9.4
        .18.v20190429; built: 2019-04-29T20:42:08.989Z; git: e1bc35120a6617ee3df052294e4
        33f3a25ce7097; jvm 1.8.0_241-b07
[clojure-agent-send-off-pool-3] INFO org.eclipse.jetty.server.handler.ContextHan
        dler - Started o.e.j.s.ServletContextHandler@59c14502{/,null,AVAILABLE}
[clojure-agent-send-off-pool-3] INFO org.eclipse.jetty.server.AbstractConnector
        - Started ServerConnector@669b2a8{HTTP/1.1,[http/1.1, h2c]}{localhost:3002}
[clojure-agent-send-off-pool-3] INFO org.eclipse.jetty.server.Server - Started @
        66258ms
[qtp1871405613-28] INFO io.pedestal.http - {:msg "GET /ola", :line 80}
[clojure-agent-send-off-pool-5] INFO org.eclipse.jetty.server.Server - jetty-9.4
        .18.v20190429; built: 2019-04-29T20:42:08.989Z; git: e1bc35120a6617ee3df052294e4
        33f3a25ce7097; jvm 1.8.0_241-b07
[clojure-agent-send-off-pool-5] INFO org.eclipse.jetty.server.handler.ContextHan
        dler - Started o.e.j.s.ServletContextHandler@4ed47a7b{/,null,AVAILABLE}
-
```





Criando um request com curl

- ✓ **Curl** é um comando disponível na maioria dos sistemas UNIX;
- ✓ **Curl** é uma abreviação de **Client URL**;
- ✓ Por meio do comando **Curl** podemos transferir dados para um servidor **HTTP** através de um request (cliente).



```
C:\Windows\system32\cmd.exe

C:\Users\Aparecido>curl --version
curl 7.28.1 (i386-pc-win32) libcurl/7.28.1 OpenSSL/1.0.1c zlib
Protocols: dict file ftp ftps gopher http https imap imaps ldap
smtp smtps telnet tftp
Features: AsynchDNS IPv6 Largefile NTLM SSL libz

C:\Users\Aparecido>_
```



Criando um request com curl

➤ `curl -i http://localhost:3002/greet`

```
Date: Tue, 28 Jul 2020 03:08:16 GMT
Content-Type: text/plain
Transfer-Encoding: chunked

Not Found
E:\Integrator_Projetos>curl -i http://localhost:3002/greet
HTTP/1.1 200 OK
Date: Tue, 28 Jul 2020 03:10:49 GMT
Strict-Transport-Security: max-age=31536000; includeSubdomains
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
X-Download-Options: noopen
X-Permitted-Cross-Domain-Policies: none
Content-Security-Policy: object-src 'none'; script-src 'unsafe-inline' 'unsafe-eval' 'strict-dynamic' https: http;;
Content-Type: text/plain
Transfer-Encoding: chunked

Hello, Pedestal...
E:\Integrator_Projetos>
```





Gerando .war da aplicação





Reconfigurando a aplicação

Arquivo project.clj

```
project.clj      core.clj

(defproject pedestal_tomcat "0.1.0-SNAPSHOT"
  :description "FIXME: write description"
  :url "http://example.com/FIXME"
  :license {:name "EPL-2.0 OR GPL-2.0-or-later WITH Classpath-exception-2.0"
            :url "https://www.eclipse.org/legal/epl-2.0/"}
  :repl-options {:init-ns user}

  :dependencies [ [org.clojure/clojure "1.10.1"]
                  [io.pedestal/pedestal.service "0.5.7"]
                  [io.pedestal/pedestal.route "0.5.7"]
                  [io.pedestal/pedestal.tomcat "0.5.8"]
                  [org.slf4j/slf4j-simple "1.7.28"] ]

  :plugins [[lein-uberwar "0.2.1"]]

  :uberwar {:handler pedestal_tomcat.core/respond-hello}

  :jvm-opts ["-Dclojure.server.myrepl={:port,5555,:accept,clojure.core.server/repl}"]

  :main ^:skip-aot api_pedestal.core

  :target-path "target/%s"

  :profiles { :uberjar { :aot :all
                        :jvm-opts ["-Dclojure.compiler.direct-linking=true"]}
              :dev { :dependencies []
                    :source-paths ["dev"]}}})
```



Reescrevendo a aplicação

Arquivo core.clj

```
core.clj

(ns pedestal_tomcat.core
  (:gen-class)
  (:require [io.pedestal.http :as http]
             [io.pedestal.http.route :as route]))

(defn respond-hello [request]
  {:status 200 :body "Hello, Pedestal com Tomcat..."})

(def routes
  (route/expand-routes
   #{" /greet" :get respond-hello :route-name :greet})))

(defn server []
  (http/create-server
   { ::http/routes routes
     ::http/type :tomcat
     ::http/port 8888}))

(defn -main
  "Aplicação para API - Tomcat com Pedestal."
  [& args]
  (println "Hello, Pedestal com Tomcat! "))
```





Gerando .war da aplicação

Na pasta do projeto > **Lein uberwar**

```
Command Prompt

28-Jul-20 11:21 AM <DIR> .
28-Jul-20 11:21 AM <DIR> ..
27-Jul-20 03:19 PM 124 .gitignore
27-Jul-20 03:19 PM 164 .hgignore
28-Jul-20 11:13 AM 108 .lein-repl-history
28-Jul-20 11:16 AM 7 .nrepl-port
27-Jul-20 03:19 PM 802 CHANGELOG.md
27-Jul-20 03:36 PM <DIR> dev
28-Jul-20 11:17 AM <DIR> doc
27-Jul-20 03:19 PM 14,652 LICENSE
28-Jul-20 01:13 PM 1,027 project.clj
27-Jul-20 03:19 PM 1,031 README.md
27-Jul-20 03:19 PM <DIR> resources
28-Jul-20 11:19 AM <DIR> src
28-Jul-20 12:52 PM <DIR> target
28-Jul-20 11:17 AM <DIR> test
28-Jul-20 11:21 AM <DIR> tmp
      8 File(s)      17,915 bytes
      9 Dir(s)  350,272,040,960 bytes free

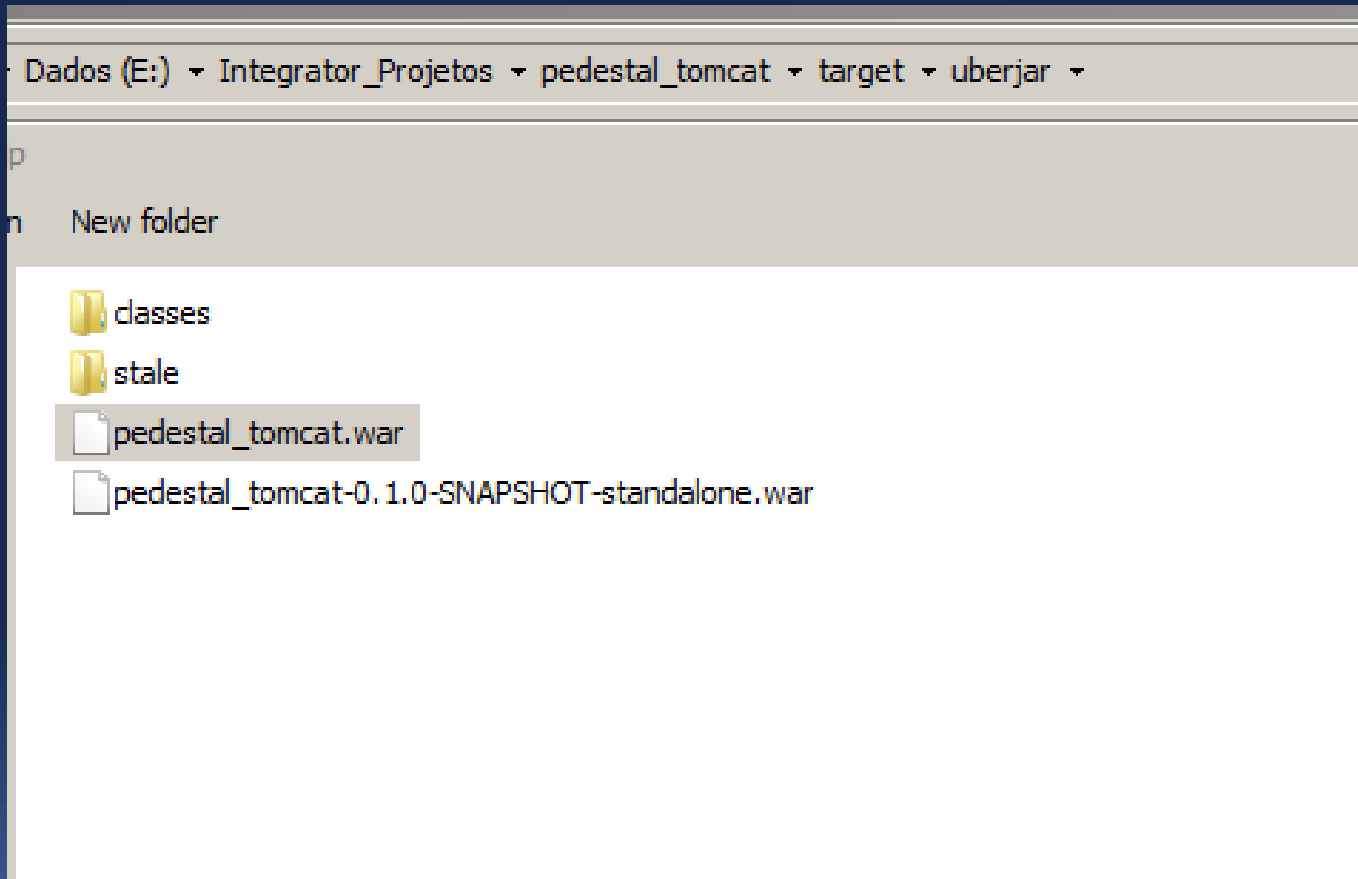
E:\Integrator_Projetos\pedestal_tomcat>atom .

E:\Integrator_Projetos\pedestal_tomcat>lein uberwar
Compiling pedestal_tomcat.core
Compiling pedestal_tomcat.core
Created E:\Integrator_Projetos\pedestal_tomcat\target\uberjar\pedestal_tomcat-0.1.0-SNAPSHOT-standalone.war
E:\Integrator_Projetos\pedestal_tomcat>
```



Renomeando .war da aplicação

Na pasta do projeto > **Lein uberwar**





Movendo .war da aplicação Para Tomcat externo

Dados (E:) ▾ apache-tomcat-9.0.34 ▾ webapps ▾

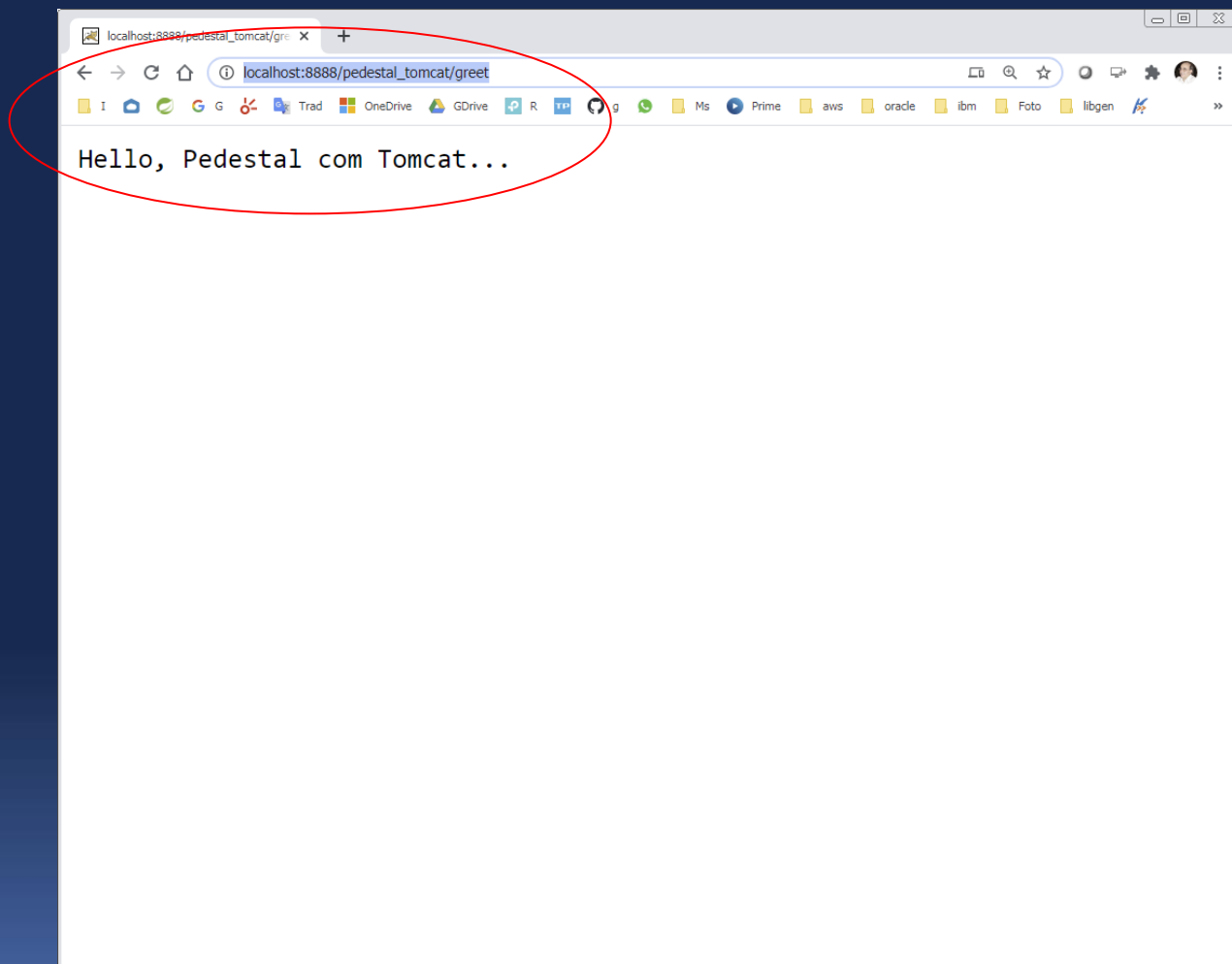
elp

rn New folder

Name ^	Date modified	Type	Size
docs	03-Apr-20 5:04 PM	File folder	
examples	03-Apr-20 5:04 PM	File folder	
hellospringboot	19-Jul-20 11:22 AM	File folder	
host-manager	03-Apr-20 5:04 PM	File folder	
listcursos	20-Jul-20 3:14 PM	File folder	
manager	03-Apr-20 5:04 PM	File folder	
ROOT	03-Apr-20 5:04 PM	File folder	
scpe	21-Jul-20 9:36 AM	File folder	
hellospringboot.war	19-Jul-20 11:06 AM	WAR File	16,109 KB
listcursos.war	20-Jul-20 3:10 PM	WAR File	16,330 KB
pedestal_tomcat.war	28-Jul-20 7:39 PM	WAR File	15,372 KB
scpe.war	17-Jul-20 8:49 PM	WAR File	11,010 KB



Aplicação rodando no Tomcat externo





Aplicação rodando na Nuvem – Integrator.com.br

