



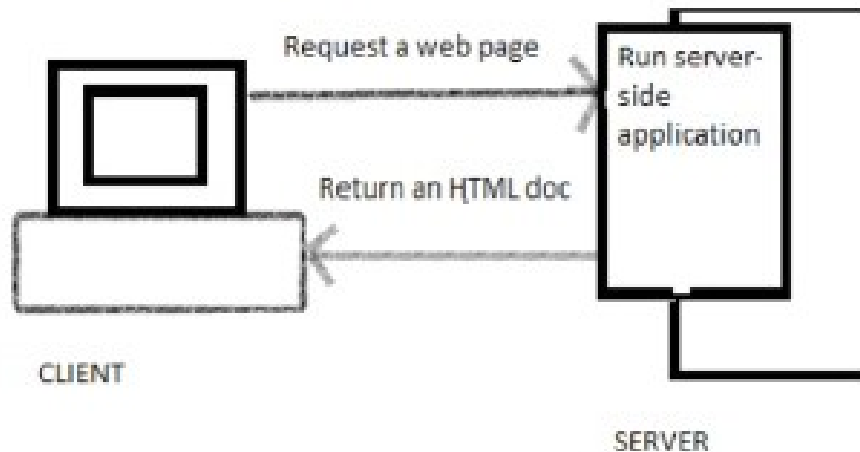
Curso RWS

Integrando JAX-RS, CDI, Beans
Validation, Vue.js, Vuetify

Fábio Henrique Barros

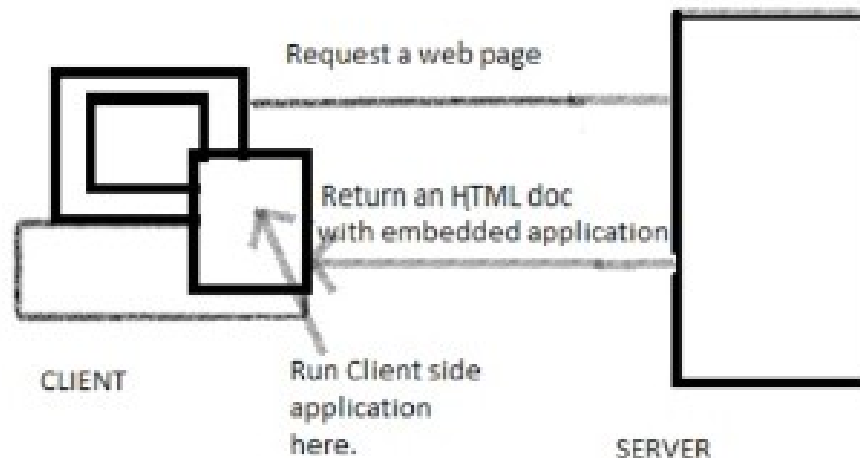
- Aplicações Rest WEB Servies
 - Introdução
 - Protocolo HTTP
 - Lado servidor da aplicação
 - REST WEB Servies
 - JAX-RS
 - CDI
 - Beans Validations
 - Lado cliente da aplicação
 - Vue.js
 - Vuetify

■ Client-side X Server-side



• Server-side

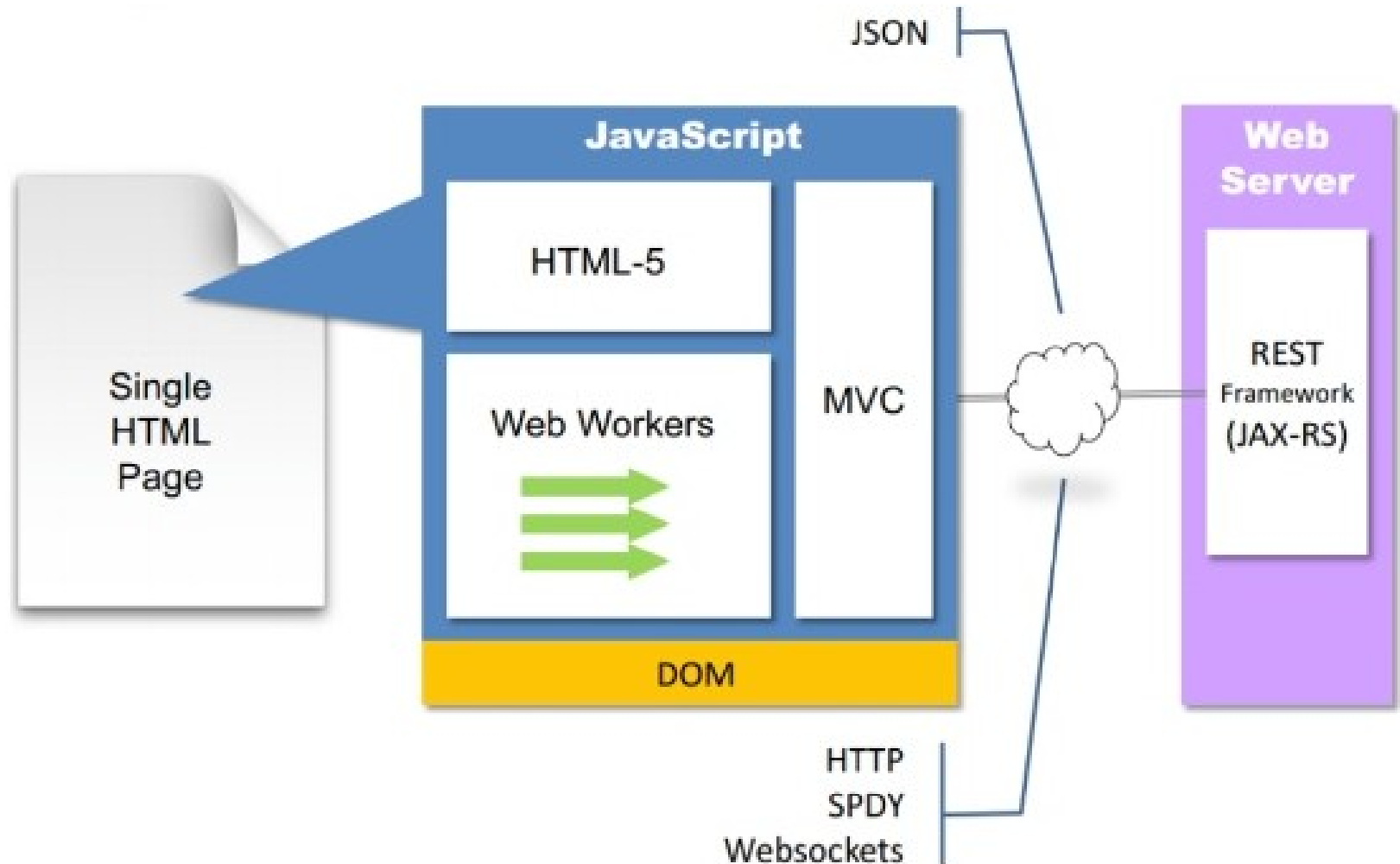
- Tudo é processado no servidor
- Mais stateful
- Fraca escalabilidade



• Client-side

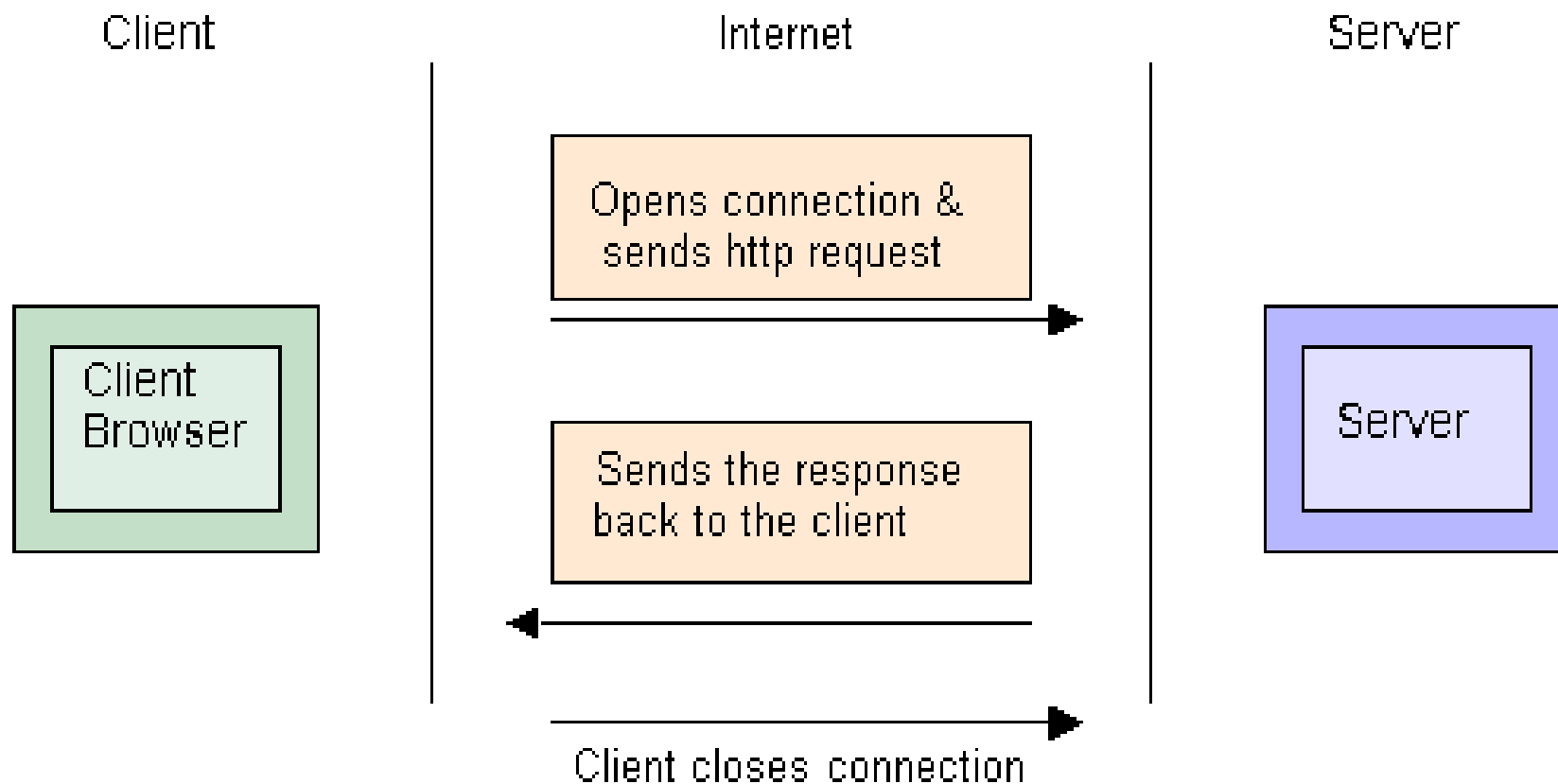
- Complexo e dinâmico
- Mais stateless
- Maior escalabilidade

- Arquitetura Rich Client

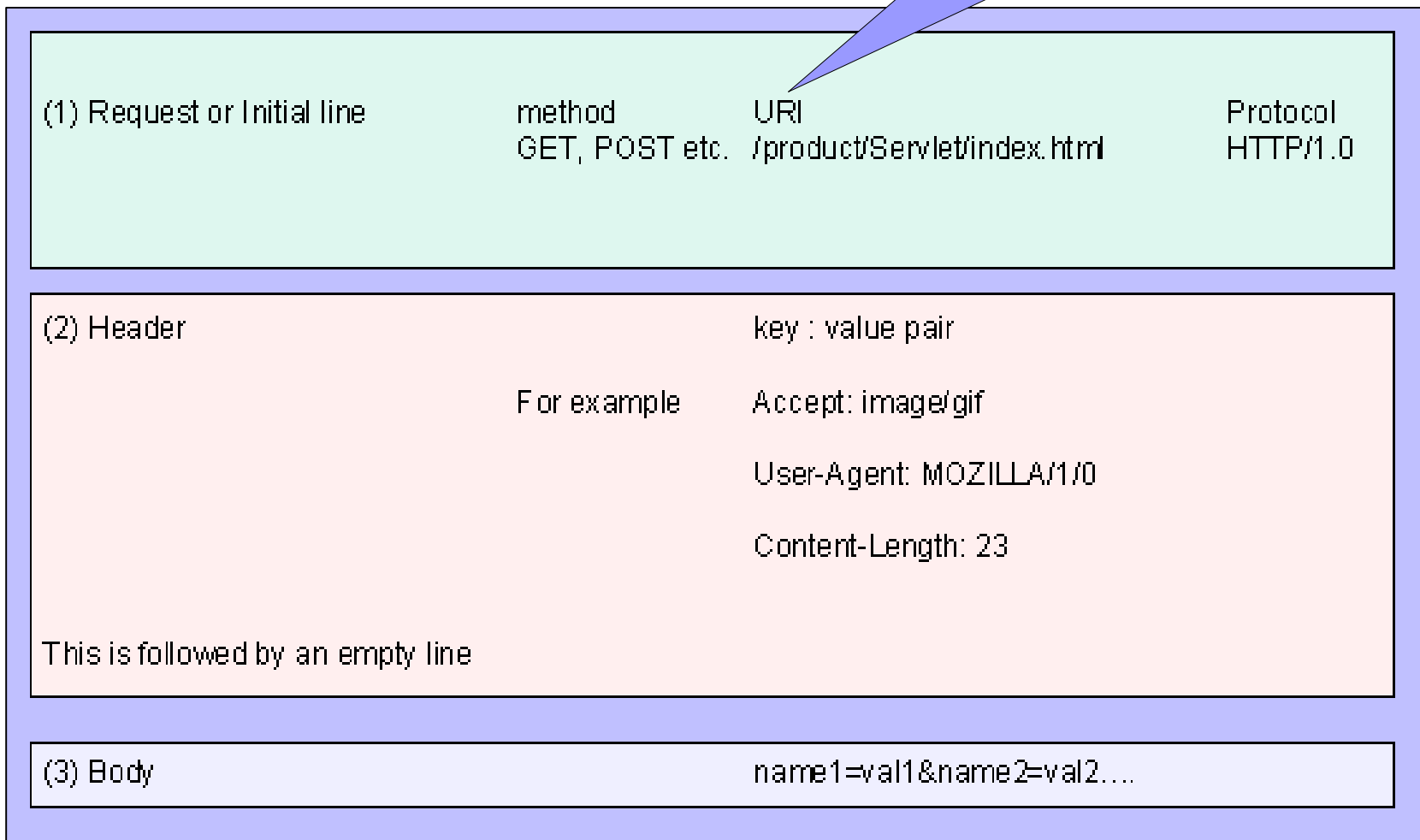


Protocolo HTTP

HTTP communication



■ Requisição HTTP



■ Métodos HTTP

■ OPTIONS (Não permite CACHE)

- Representa um pedido de informações sobre as opções de comunicação disponíveis na cadeia requisição/resposta identificado pela URI.
- Permite ao cliente determinar as opções e/ou requisitos associados a um recurso(s) de um servidor, sem implicar numa ação sobre estes recursos.
- Exemplo:
 - OPTIONS http://localhost/usuarios/1
 - 200 OK
Allow: HEAD,GET,PUT,DELETE,OPTIONS
Content-Type: application/json

■ Métodos HTTP

■ GET (Permite CACHE)

- Recupera todas as informações em forma de uma entidade, identificadas na URI.
- Se a URI refere-se à uma entidade de dados, deverá retornar a entidade como resposta. Se refere-se à um documento, deverá retornar o próprio documento.

■ Exemplo:

- GET `http://localhost/usuarios/1`

- 200 OK

Content-Type: application/json

```
{"id":1,"cpf":"11111111111","nome":"Pedro de Alcantara",
```

```
"email":"pedro.alcantara@gmail.com",
```

```
"senha":"teste123","data":"12/10/1798"}
```


■ Métodos HTTP

- HEAD (Permite CACHE, mas com atenção)
 - Idêntico ao GET, mas o servidor não deve retornar o corpo da resposta. Os cabeçalhos HTTP da resposta devem ser iguais.
 - Frequentemente utilizado para testes de validade, acessibilidade e modificação recente.
 - Exemplo:
 - HEAD http://localhost/usuarios/1
 - 200 OK
Content-Type: application/json
Last-Modified: Wed, 25 Feb 2016 22:37:23 GMT
ETag: "1450013-6514-e905eec0"
Content-Length: 25876

■ Métodos HTTP

■ POST (Não permite CACHE)

- Permite postar novas informações para o servidor. Ex:

- Inserir
- Complementar
- Atualizar
- Publicar

- Exemplo:

- POST http://localhost/usuarios
- 201 Created
Content-Type: application/json
Location: /usuarios/4
Content-Length: 14

■ Métodos HTTP

■ PUT (Não permite CACHE)

- Permite que a entidade especificada na URI seja atualizada.
- Caso a entidade não exista, uma nova deverá ser criada.
- Exemplo:
 - PUT `http://localhost/usuarios/4`
 - 200 OK
Content-Type: application/json
Content-Length: 14

■ Métodos HTTP

■ DELETE (Não permite CACHE)

- Exclui a entidade identificada na URI.
- Resposta de sucesso deve ser:
 - 200 (OK): Entidade excluída e retornada na resposta
 - 202 (Accepted): Exclusão foi aceita mas agendada
 - 204 (No Content): Entidade excluída e a resposta é vazia
- Exemplo:
 - DELETE http://localhost/usuarios/4
 - 200 OK
Content-Type: application/json
{ "id":1, "cpf":"11111111111", "nome":"Pedro de Alcantara", "email":"pedro.alcantara@gmail.com", "senha":"teste123", "data":"12/10/1798" }

■ Métodos HTTP

■ TRACE (Não permite CACHE)

- Executa um teste de ECHO da requisição.

- Exemplo:

- TRACE http://localhost/usuarios

User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:48.0) Gecko/20100101 Firefox/48.0

- 200 OK

Date: Mon, 27 Jul 2016 12:28:53 GMT

Server: Apache/2.2.14 (Win32)

Content-Type: message/http

Content-Length: 39

TRACE http://localhost/usuarios

User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:48.0) Gecko/20100101 Firefox/48.0

■ Resposta HTTP

(1) Response or status line	Protocol HTTP/1.0	Status Code 200	Description OK
(2) Header	key : value [content-Type : text/html]		
(3) Body	<HTML> -- -- </HTML>		

- Códigos de resposta HTTP
 - Devem estar na resposta gerada pelo servidor.
 - O cliente deve tomar ações de acordo com o código da resposta.
 - Podemos criar nossos próprios códigos.
- Veja mais em:
 - <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status>
 - <https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>

■ Informational 1xx

■ **100 Continue**

- O cliente deve continuar seu pedido. Resposta provisória que informa ao cliente que a parte da solicitação foi recebida e não foi rejeitada. O cliente deve continuar enviando o restante do pedido. O servidor deve enviar uma resposta final depois que a solicitação foi concluída. (Upload de arquivos)

■ **101 Switching Protocols**

- O servidor entende e está disposto a cumprir o pedido do cliente, mas solicita a mudança de protocolo através do campo de cabeçalho Upgrade.

■ Successful 2xx

■ 200 OK

- O pedido foi bem sucedido.

■ 201 Created

- O pedido foi cumprido e resultou em um novo recurso.

■ 202 Accepted

- O pedido foi aceito para processamento, mas o processamento ainda não foi concluído.

■ 204 No Content

- O servidor cumpriu o pedido, mas não precisa retornar uma entidade do corpo da resposta. Pode retornar metadados atualizados.

■ Redirection 3xx

■ **301 Moved Permanently**

- O recurso solicitado recebeu uma nova URI permanente. Futuras requisições a este recurso deverá usar a nova URI retornada.

■ **302 Found**

- O recurso solicitado reside temporariamente em uma URI diferente. O redirecionamento pode ser alterado ocasionalmente

■ Redirection 3xx (cont.)

■ 304 Not Modified

- Se o cliente executar um GET e o acesso for permitido, mas o documento não foi modificado, o servidor deverá responder esse código de status.

■ 307 Temporary Redirect

- O recurso solicitado reside temporariamente em uma URI diferente.

■ Client Error 4xx

■ **400 Bad Request**

- A solicitação não pôde ser entendida pelo servidor devido à sintaxe malformada.

■ **401 Unauthorized**

- O pedido requer autenticação do usuário.

■ **403 Forbidden**

- O pedido requer autorização do usuário.

■ Client Error 4xx (cont.)

■ **404 Not Found**

- O servidor não encontrou nada para a URI solicitada.

■ **405 Method Not Allowed**

- O método especificado na requisição não é permitido para o recurso identificado pela URI.

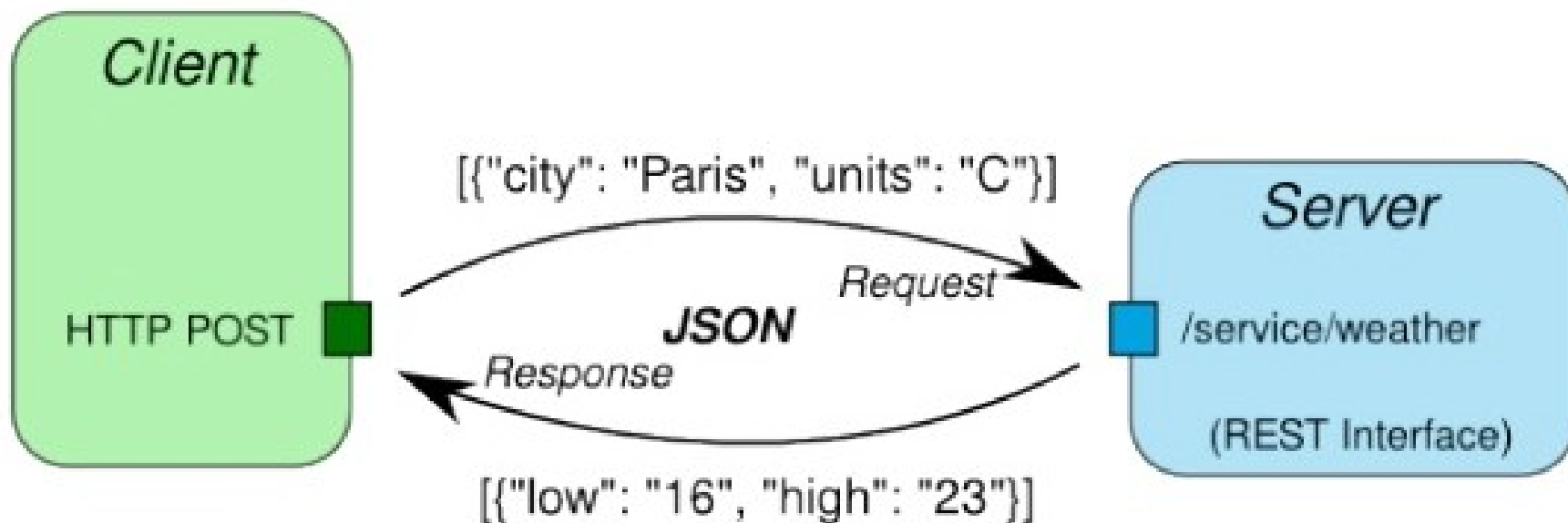
■ **406 Not Acceptable**

- O recurso identificado pela solicitação não é capaz de gerar entidades de resposta de acordo com os cabeçalhos enviados na requisição.

- Aplicações Rest WEB Servies (4ª Geração)
 - Representational State Transfer (REST) é um estilo de arquitetura baseado em:
 - Interface uniforme
 - Desempenho
 - Escalabilidade
 - Manutenibilidade
 - No estilo REST, dados e funcionalidade são considerados recursos e são acessados através de Uniform Resource Identifier (URI).
 - Arquitetura cliente/servidor projetada para usar um protocolo stateless, normalmente HTTP.

- Arquitetura REST

JSON / REST / HTTP



- Princípios REST

- **Identificação de recursos através de URI:**

- Um serviço RESTful expõe um conjunto de recursos que identificam os alvos da interação com seus clientes.
 - Os recursos são identificados por URI's, que proporcionam um endereçamento global para recursos e descoberta de serviços.

- Princípios REST

- **Interface uniforme:**

- Os recursos são manipulados usando um conjunto fixo de quatro operações:
 - GET: recupera o estado atual de um recurso.
 - POST: cria um novo recurso.
 - PUT: transfere um novo estado para um recurso.
 - DELETE: exclui recurso.

- Princípios REST

- **Mensagens auto-descritivas:**

- Os recursos são dissociados da sua representação para que o seu conteúdo possa ser acessado em uma variedade de formatos, como HTML, XML, texto simples, PDF, JPEG, JSON, e outros.
 - Metadados podem ser usados, para controlar o cache, detectar erros de transmissão, negociar o formato de representação adequada, realizar a autenticação ou controle de acesso.

■ Princípios REST

■ **Interações com estado através de links:**

- Cada interação com um recurso é stateless; ou seja, as mensagens de requisição são auto-suficientes.
- Existem várias técnicas para a troca de estado, como reescrita de URL, cookies e campos ocultos de formulário.
- Estado pode ser incorporado em mensagens de resposta, para apontar para um futuro estado válido de interação.

- Suporte a REST em Java
- API padrão
 - Programação declarativa
 - Abstrações para implementação no server e client
 - Serviços implementados via POJO
 - Plugável e extensível
 - Providers, filters, interceptors, validators
 - Suporta processamento assíncrono
 - Integrado com as tecnologias do Java EE
- Configuração via anotações
 - @Path, @GET, @POST, @PUT, @DELETE, @PathParam, @QueryParam, @Produces, @Consumes, etc

■ Annotations:

■ @Path

- Define a URI para ser utilizada pelo endpoint.

■ @GET, @POST, @PUT, @DELETE, @HEAD

- Determina acesso ao serviço via método HTTP.

■ @PathParam

- Define o mapeamento do valor informado na URI para um determinado parâmetro de método.

■ @QueryParam

- Define o mapeamento do valor informado na query string para um determinado parâmetro de método.

■ Annotations:

■ @Consumes

- Define um determinado MIME type para recebimento de dados pelo serviço.

■ @Produces

- Define um determinado MIME type para envio de dados pelo serviço.

■ @Provider

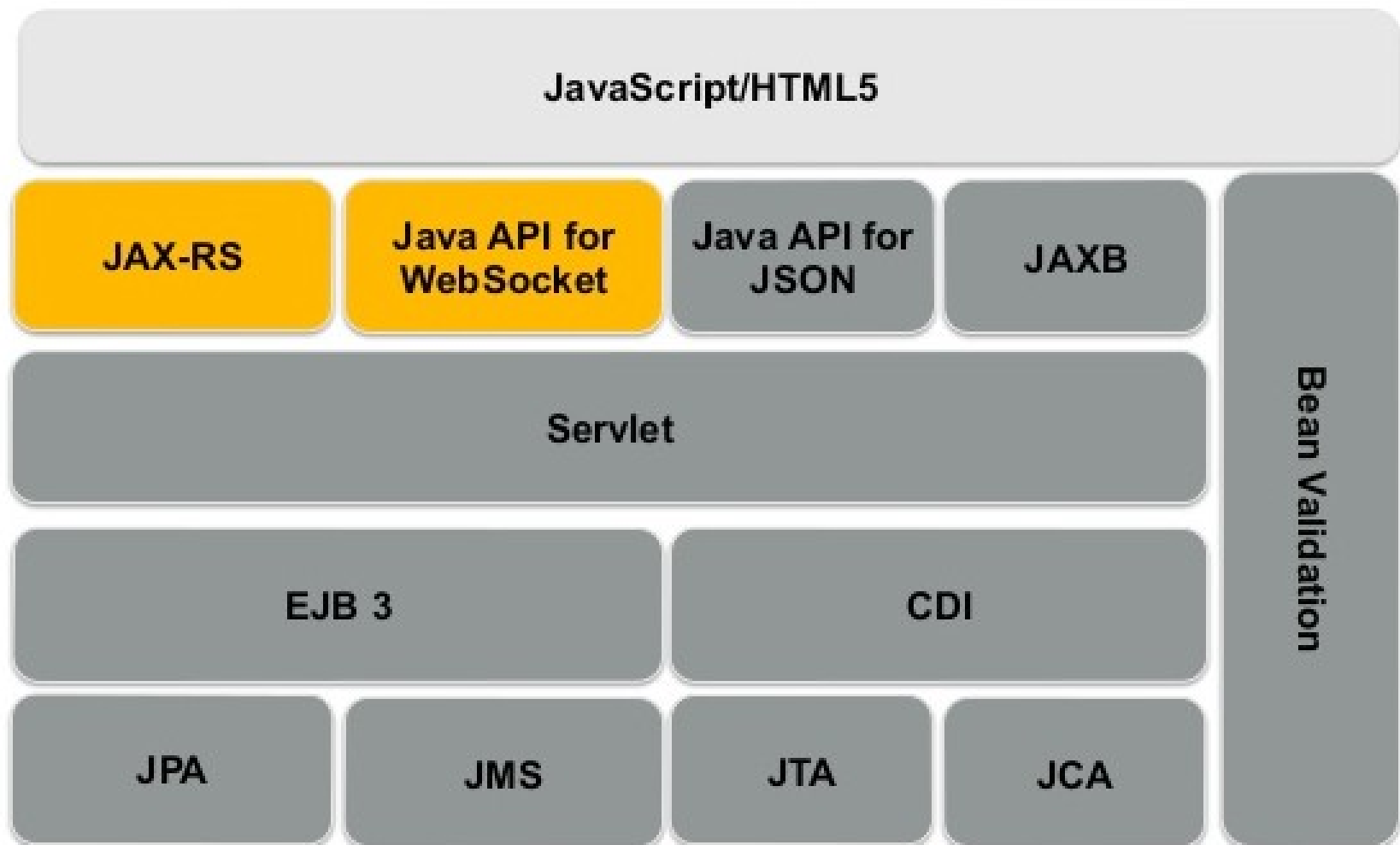
- Define um determinado componente para auxiliar no JAX-RS runtime.

■ @ApplicationPath

- Determina o root path de uma aplicação JAX-RS.

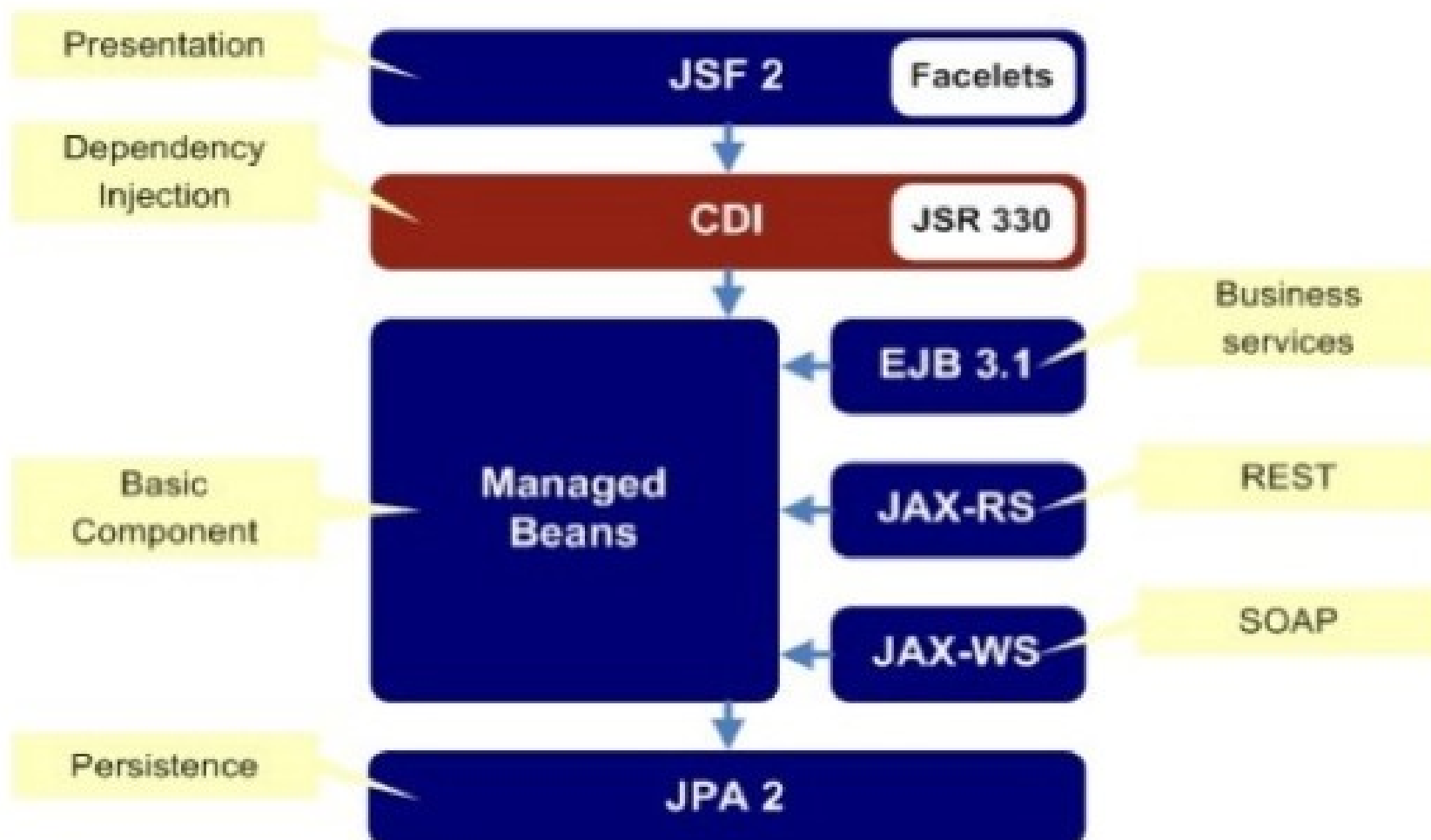
- Annotations:
 - @QueryParam e @DefaultValue
 - Extraem da query string (?nome=valor&nome=valor)
 - @FormParam
 - Extrai do form (applicaton/x-www-form-urlencoded)
 - @CookieParam
 - Extrai de cookies (pares nome=valor)
 - @HeaderParam
 - Extrai de cabeçalhos HTTP
 - @MatrixParam
 - Extrai de segmentos de URL

- JEE + Javascript



- CDI API, que implementa a injeção de dependência e contextos.
- Parte da especificação do Java EE 6 (JSR 299).
- Uma alternativa para os frameworks de injeção de dependência como Spring ou Google Guice.
- Um sucessor do JBOSS Seam Framework.

■ Integrações no JEE



- O QUE É INJEÇÃO DE DEPENDÊNCIA?
 - Basicamente instanciação de objetos.
 - Contêiner:
 - Objetos são criados por um contêiner que os associa a um contexto e gerencia seu ciclo de vida.
 - As referências entre instâncias também são inicializadas pelo container e são injetados no objeto.
 - A configuração de criação do objeto e a injeção é feita utilizando:
 - Anotações.
 - Configuração de XML.
 - Código Java.

- PARA QUE INJEÇÃO DE DEPENDÊNCIA?
 - Manter o baixo acoplamento entre os componentes.
 - Facilidade na criação de testes.
 - Código mais limpo.
 - Isto leva a um maior grau de flexibilidade para diferentes configurações de um aplicativo.
 - Por exemplo, em cenários de teste, ambientes de integração e a reutilização em outras aplicações ou contextos.

- O que são contextos?
 - Os contextos determinam o ciclo de vida dos componentes, na CDI (duração e visibilidade) dos objetos.
 - O desenvolvedor configura o escopo usando anotações:
 - @ApplicatonScoped (aplicação).
 - @SessionScoped (sessão do usuário).
 - @ConversatonScoped (conversação).
 - @RequestScoped (http request).
 - @Dependent (depende do ciclo de vida do componente referenciado).

Implementando a API (consulta)

■ Usuario.java

```
public class Usuario extends BaseModel implements Serializable {  
  
    private static final long serialVersionUID = -3064306490724801147L;  
  
    private String cpf;  
    private String nome;  
    private String email;  
    private String senha;  
    private Date data;  
  
    public Usuario() {  
        super();  
    }  
  
    public Usuario(Long id) {  
        this();  
        setId(id);  
    }  
  
    /* Getters e Setters */  
}
```

Implementando a API (consulta)

■ UsuarioBC.java

@ApplicationScoped

```
public class UsuarioBC {
```

@Inject

```
private Repositorio repositorio;
```

@PostConstruct

```
public void inicializar() {
```

```
    Calendar data = Calendar.getInstance();
```

```
    Usuario usuario = new Usuario();
```

```
    usuario.setNome("Pedro de Alcantara");
```

```
    usuario.setEmail("pedro.alcantara@gmail.com");
```

```
    usuario.setSenha("teste123");
```

```
    usuario.setCpf("11111111111");
```

```
    data.set(1798, 9, 12);
```

```
    usuario.setData(data.getTime());
```

```
    repositorio.inserir(usuario);
```

```
...
```

Implementando a API (consulta)

■ UsuarioBC.java

```
...  
    usuario = new Usuario();  
    usuario.setNome("Santos Dumont");  
    usuario.setEmail("santos.dumont@gmail.com");  
    usuario.setSenha("teste123");  
    usuario.setCpf("2222222222");  
    data.set(1873, 6, 20);  
    usuario.setData(data.getTime());  
    repositorio.inserir(usuario);  
  
    usuario = new Usuario();  
    usuario.setNome("Isabel de Braganca");  
    usuario.setEmail("maria@gmail.com");  
    usuario.setSenha("teste123");  
    usuario.setCpf("3333333333");  
    data.set(1846, 6, 29);  
    usuario.setData(data.getTime());  
    repositorio.inserir(usuario);  
}
```


Implementando a API (consulta)

■ UsuarioBC.java

...

```
public List<Usuario> selecionar() {  
    return repositorio.selecionar(Usuario.class);  
}  
  
public Usuario selecionar(Long id) throws  
    UsuarioNaoEncontradoException {  
  
    Usuario usuario = repositorio.selecionar(Usuario.class, id);  
    if (usuario == null) {  
        throw new UsuarioNaoEncontradoException();  
    }  
    return usuario;  
}  
  
}
```

Implementando a API (consulta)

■ UsuarioRS.java

```
@Path("usuarios")
public class UsuariosRS {
    @Inject
    private UsuarioBC usuarioBC;

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Usuario> selecionar() {
        return usuarioBC.selecionar();
    }

    @GET
    @Path("{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public Usuario selecionar(@PathParam("id") Long id) {
        try {
            return usuarioBC.selecionar(id);
        } catch (UsuarioNaoEncontradoException e) {
            throw new NotFoundException();
        }
    }
}
```

Implementando a API (consulta)

■ Api.java

```
@ApplicationPath("api")  
public class Api extends Application {  
  
}
```

■ Teste:

- <http://localhost:8080/cursorws/api/usuarios>
- <http://localhost:8080/cursorws/api/usuarios/1>

- Hypertext Markup Language (Linguagem de Marcação de Hipertexo)
 - `<a/><p/><div/>`
 - `<table/>
`
- Linguagem Interpretada (Browser).
- Define o formato do documento e as ligações com outros documentos.
- Em suma: “O HTML é uma linguagem para publicação de conteúdo na Web (texto, imagem, vídeo, áudio, etc.)”.



- Desenvolvido por Tim Berners-Lee.
- Entre 1993 e 1995, o HTML ganhou as versões HTML+, HTML2.0 e HTML3.0.
 - Ainda não era um padrão
- Em 1997 o W3C criou a versão 3.2.
 - A partir daí HTML se tornou um padrão.
- Em 1999 saiu a especificação do HTML4.
- Em 2004 o W3C lança o xHTML1.0 para reformular o HTML4.

- Mozilla, Opera, Safari, criam o grupo WHATWG (Web Hypertext Application Technology Working Group) e desenvolvem HTML independente do W3C.
- Em 2007 surge o xHTML2.0 com influências do WHATWG.
- Em 2008 o W3C abandona o xHTML2.0 e anuncia oficialmente o HTML5.0.
- Em 2012 sai o primeiro draft da especificação do HTML5.

■ HTML5

- Necessidade de criar elementos semânticos.
- Cria novas tags e muda a função de outras.
- Padrão para criação de sessões comuns como rodapé, cabeçalho, sidebar e menus.
- Muda a forma de escrever código e organizar a informação:
 - Mais semântica e menos código;
 - Mais interatividade sem plugins;
 - Código interoperável, pronto para futuros dispositivos;
 - Facilita a reutilização da informação de diversas formas.

- Não faz parte da especificação HTML5, mas é uma parte integral do desenvolvimento WEB.
- Está sendo desenvolvido em conjunto com HTML5 e fornece muitos novos estilos que fazem com que as páginas Web funcionem pareçam muito melhor que antes.
- Coisas que estavam apenas disponíveis no Photoshop, como gradientes e sombras, agora são facilmente adicionadas via estilo.



- O uso dessas novas características gráficas tornarão nossas aplicações mais modernas.
- Definidos em arquivos separados ou na própria página HTML.
- Exemplo:

```
.erro {  
    color: red;  
    font-weight: bold;  
    border: 1px solid red;  
    border-radius: 4px;  
}
```

- JavaScript (JS) é uma linguagem de programação interpretada criada em 1995 como uma extensão do HTML.
- O JS oferece recursos que faltam no HTML, permitindo a criação de páginas interativas e dinâmicas, que são interpretadas localmente.
- O JS é hoje executado em todos os browsers, e está sendo amplamente disseminado para outras plataformas, como servidores, desktops e mobile.



- JavaScript não é Java:
 - JS frequentemente é confundida com a linguagem Java, provavelmente devido à semelhança do nome.
 - Apesar de possuir sintaxe parecida, não tem nada a ver com Java!
 - O nome "script", que quer dizer roteiro, já indica que se trata de uma linguagem interpretada.

- Brendan Eich utilizou várias linguagens:

- Java

- Sintaxe
 - Algumas convenções
 - Date e Math

- Scheme

- Lambda
 - Closure
 - Tipagem fraca

- Self

- Herança baseada em protótipos
 - Objetos dinâmicos

- Perl

- Expressões Regulares

- O que podemos fazer com JS?
 - Realizar operações matemáticas e de computação.
 - Gerar documentos com aparência definida na hora da visualização, com base em informações do cliente.
 - Abrir janelas, trocar e manipular informações como o histórico, barra de estado, plug-ins, etc.
 - Interagir com o conteúdo do documento, alterando propriedades da página.
 - Interagir com o usuário através de eventos.

■ Formas de usar o JS

- Dentro de blocos HTML `<SCRIPT> ... </SCRIPT>` em várias partes da página.
 - Para definir funções usadas pela página, gerar HTML em novas páginas ou alterar o procedimento de interpretação do HTML.
- Em um arquivo externo, importado pela página.
 - Para definir funções que serão usadas por várias páginas em um site.
- Dentro de descritores HTML sensíveis a eventos.
 - Para tratar eventos do usuário em elementos da página durante a exibição.

■ JSON

- JavaScript Object Notation
 - {id:123, cidade:"Paris", voos:["M344","J919"]}
- Pode ser codificado diretamente em String e processado com split(), substring(), indexOf()...
- Java EE 7 disponibiliza um API para construir objetos JSON e para converter JSON em mapas
 - APIs de baixo nível (não é mapeamento objeto-JSON)
- Outras implementações fazem mapeamento objeto-JSON automático (não são parte do JEE)
 - MOXy, Jettison, Jersey, Jackson, etc.

Tríade do desenvolvimento WEB

JS



HTML



CSS



Vue.js



Vue.js



Vuetify

Implementando o Cliente (consulta)

■ index.html

```
<!DOCTYPE html>
<html>

<head>
  <title>Curso RWS - Index</title>

  <meta charset="utf-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1, maximum-
scale=1, user-scalable=no, minimal-ui">

  <link href="https://fonts.googleapis.com/css?
family=Roboto:100,300,400,500,700,900|Material+Icons" rel="stylesheet">
  <link href="https://unpkg.com/vuetify/dist/vuetify.min.css" rel="stylesheet">

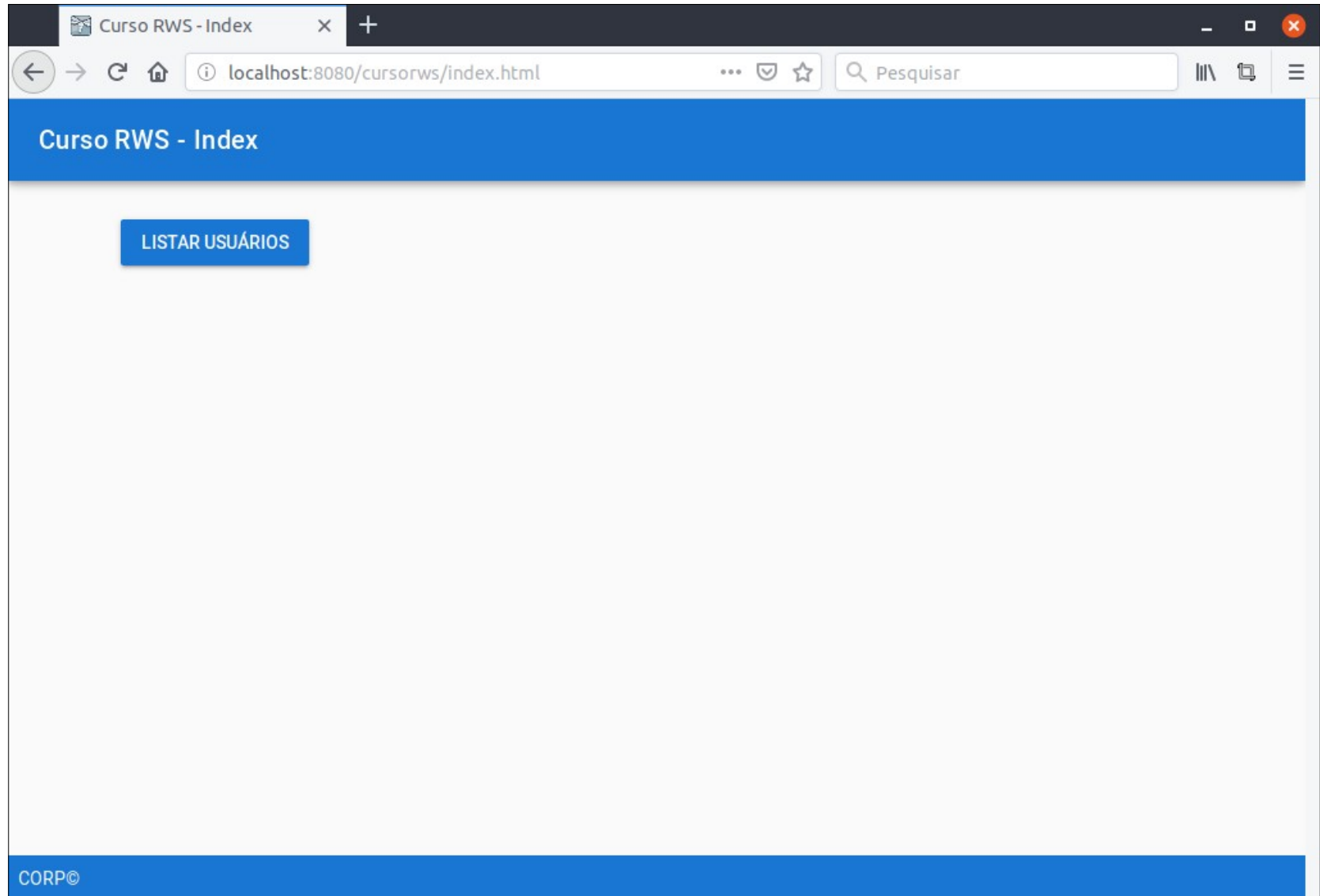
  <script src="https://unpkg.com/vue/dist/vue.js"></script>
  <script src="https://unpkg.com/vuetify/dist/vuetify.js"></script>
</head>
...
```

Implementando o Cliente (consulta)

■ index.html

```
...  
<body>  
  <v-app id="app">  
    <v-toolbar dark app color="primary">  
      <v-toolbar-title>Curso RWS - Index</v-toolbar-title>  
    </v-toolbar>  
    <v-content>  
      <v-container>  
        <v-btn color="primary" href="usuarios-listar.html">  
          Listar Usuários  
        </v-btn>  
      </v-container>  
    </v-content>  
    <v-footer dark app color="primary" class="pa-2">  
      CORP&copy;  
    </v-footer>  
  </v-app>  
  <script>  
    var app = new Vue({  
      el : '#app'  
    });  
  </script>  
</body>  
</html>
```

Implementando o Cliente (consulta)



Implementando o Cliente (consulta)

■ /js/usuarios-service.js

```
class UsuariosService {
  constructor() {
    this.axios = axios.create({ baseURL: 'api/usuarios' });
  }
  request(method, url, data) {
    return this.axios[method](url, data)
      .then(response => {
        return response.data;
      })
      .catch(error => {
        console.error(
          '[ERROR: UsuariosService] ' + method + ' ' + url, error);
        return Promise.reject(error);
      });
  }
  seleccionar(id) {
    return this.request('get', '/' + id);
  }
  seleccionarTodos() {
    return this.request('get');
  }
}

var usuariosService = new UsuariosService();
```

Implementando o Cliente (consulta)

■ usuarios-listar.html

```
<!DOCTYPE html>
<html>

<head>
  <title>Curso RWS - Listagem de Usuários</title>

  <meta charset="utf-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1, maximum-
scale=1, user-scalable=no, minimal-ui">

  <link href="https://fonts.googleapis.com/css?
family=Roboto:100,300,400,500,700,900|Material+Icons" rel="stylesheet">
  <link href="https://unpkg.com/vuetify/dist/vuetify.min.css" rel="stylesheet">
  <script src="https://unpkg.com/axios@0.18.0/dist/axios.js"></script>
  <script src="https://unpkg.com/vue/dist/vue.js"></script>
  <script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>
  <script src="https://unpkg.com/vuetify/dist/vuetify.js"></script>
</head>
...
```


Implementando o Cliente (consulta)

■ usuarios-listar.html

```
...
<body>
  <v-app id="app">
    <v-toolbar dark app color="primary">
      <v-btn flat icon href="index.html">
        <v-icon>keyboard_arrow_left</v-icon>
      </v-btn>
      <v-toolbar-title>Curso RWS - Listagem de Usuários</v-toolbar-title>
    </v-toolbar>
    <v-content>
      <v-alert v-model="mensagem.exibir" :type="mensagem.tipo" dismissible>
        {{ mensagem.texto }}
      </v-alert>
      <v-container>
        <v-data-table class="elevation-3" :headers="headers" :items="usuarios">
          <template slot="no-data">
            Nenhum usuário carregado
          </template>
          <template slot="items" slot-scope="usuario">
            <td>
              {{ usuario.item.cpf }}
            </td>
          </template>
        </v-data-table>
      </v-container>
    </v-content>
  </v-app>
</body>
...
```

Implementando o Cliente (consulta)

■ usuarios-listar.html

```
...  
    <td>{{ usuario.item.nome }}</td>  
    <td>{{ usuario.item.email }}</td>  
    <td>{{ usuario.item.data }}</td>  
</template>  
<template slot="actions-append">  
    <v-btn flat icon @click="carregar">  
        <v-icon>cached</v-icon>  
    </v-btn>  
</template>  
</v-data-table>  
</v-container>  
</v-content>  
<v-footer dark app color="primary" class="pa-2">  
    CORP&copy;  
</v-footer>  
</v-app>  
<script src="js/usuarios-service.js"></script>  
<script src="js/usuarios-listar.js"></script>  
</body>  
</html>
```

Implementando o Cliente (consulta)

■ /js/usuarios-listar.js

```
var app = new Vue({
  el: '#app',

  data() {
    return {
      mensagem: {},
      headers: [
        { text: 'CPF', sortable: true, value: 'cpf' },
        { text: 'Nome', sortable: true, value: 'nome' },
        { text: 'E-mail', sortable: true, value: 'email' },
        { text: 'Data Nascimento', sortable: true, value: 'data' },
      ],
      usuarios: [],
    }
  },

  mounted() {
    this.carregar();
  },

  ...
```

Implementando o Cliente (consulta)

■ /js/usuarios-listar.js

```
...
methods: {
  carregar() {
    this.limparMensagem();
    usuariosService.selecionarTodos()
      .then(usuarios => {
        this.usuarios = usuarios;
      })
      .catch(error => {
        this.usuarios = [];
        this.exibirMensagem('error', 'Erro inesperado.');
```

```
      });
  },

  exibirMensagem(tipo, texto) {
    this.mensagem = { tipo, texto, exibir: true };
  },
```

```
  limparMensagem() {
    this.mensagem = { tipo: 'info', texto: '', exibir: false };
  },
```

```
};
});
```

Implementando o Cliente (consulta)

Curso RWS - Listagem de x +

localhost:8080/cursorws/usuarios-listar.html

Pesquisar

< Curso RWS - Listagem de Usuários

CPF ↑	Nome	E-mail	Data Nascimento
11111111111	Pedro de Alcantara	pedro.alcantara@gmail.com	1798-10-12T18:47:44.348Z[UTC]
22222222222	Santos Dumont	santos.dumont@gmail.com	1873-07-20T18:47:44.348Z[UTC]
33333333333	Isabel de Braganca	maria@gmail.com	1846-07-29T18:47:44.348Z[UTC]

Rows per page: 5 1-3 of 3 < > ↺

CORP©

Implementando o Cliente (consulta)

- Corrigindo a formatação da data
 - JAX-RS usa o JSONB como serializador de objetos.
 - Todas as facilidades do JSONB estão disponíveis.
 - Parar formatar a data devemos implementar um JsonbAdapter:

```
public interface JsonbAdapter<Original, Adapted> {  
    Adapted adaptToJson(Original obj) throws Exception;  
    Original adaptFromJson(Adapted obj) throws Exception;  
}
```

Implementando o Cliente (consulta)

■ DateAdapter.java

```
public class DateAdapter implements JsonbAdapter<Date, String> {

    private static final SimpleDateFormat sdf =
        new SimpleDateFormat("dd/MM/yyyy");

    @Override
    public String adaptToJson(Date value) throws Exception {
        if (value == null) {
            return "";
        }
        return sdf.format(value);
    }

    @Override
    public Date adaptFromJson(String value) throws Exception {
        if (value == null || value.trim().equals("")) {
            return null;
        }
        return sdf.parse(value);
    }
}
```

Implementando o Cliente (consulta)

- Use a anotação `@JsonbTypeAdapter` para ativar o adapter à propriedade desejada.
- `Usuario.java`

```
public class Usuario extends BaseModel implements Serializable {  
    ...  
  
    @JsonbTypeAdapter(DateAdapter.class)  
    private Date data;  
  
    ...  
}
```


Implementando o Cliente (consulta)

Curso RWS - Listagem de x +

localhost:8080/cursorws/usuarios-listar.html

Pesquisar

< Curso RWS - Listagem de Usuários

CPF ↑	Nome	E-mail	Data Nascimento
11111111111	Pedro de Alcantara	pedro.alcantara@gmail.com	12/10/1798
22222222222	Santos Dumont	santos.dumont@gmail.com	20/07/1873
33333333333	Isabel de Braganca	maria@gmail.com	29/07/1846

Rows per page: 5 1-3 of 3 < > ↺

CORP©

Implementando a API (inclusão)

■ UsuariosBC.java

@ApplicationScoped

```
public class UsuarioBC {
```

```
    ...
```

```
    public Long inserir(Usuario usuario) {  
        return repositorio.inserir(usuario);  
    }
```

```
}
```

Implementando a API (inclusão)

■ UsuariosRS.java

```
@Path("usuarios")
public class UsuariosRS {
    ...
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Response inserir(Usuario body) {
        Long id = usuarioBC.inserir(body);
        String url = "/api/usuarios/" + id;
        return Response
            .status(Status.CREATED)
            .header("Location", url)
            .entity(id)
            .build();
    }
}
```

Implementando a API (inclusão)

■ usuarios-editar.html

```
<!DOCTYPE html>
<html>

<head>
  <title>Curso RWS - Cadastro de Usuários</title>

  <meta charset="utf-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1, maximum-
scale=1, user-scalable=no, minimal-ui">

  <link href="https://fonts.googleapis.com/css?
family=Roboto:100,300,400,500,700,900|Material+Icons" rel="stylesheet">
  <link href="https://unpkg.com/vuetify/dist/vuetify.min.css" rel="stylesheet">
  <script src="https://unpkg.com/axios@0.18.0/dist/axios.js"></script>
  <script src="https://unpkg.com/vue/dist/vue.js"></script>
  <script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>
  <script src="https://unpkg.com/vuetify/dist/vuetify.js"></script>
</head>
```

...

Implementando a API (inclusão)

■ usuarios-editar.html

```
...  
<body>  
  <v-app id="app">  
    <v-toolbar dark app color="primary">  
      <v-btn flat icon href="usuarios-listar.html">  
        <v-icon>keyboard_arrow_left</v-icon>  
      </v-btn>  
      <v-toolbar-title>Curso RWS - Cadastro de Usuários</v-toolbar-title>  
    </v-toolbar>  
    <v-content>  
      <v-alert v-model="mensagem.exibir" :type="mensagem.tipo" dismissible>  
        {{ mensagem.texto }}  
      </v-alert>  
      <v-container>  
        <v-layout row wrap>  
          <v-flex xs2>  
            <v-text-field type="text" label="CPF:"  
              v-model="usuario.cpf" :error-messages="erros.cpf" />  
          </v-flex>  
          <v-flex xs12>  
            <v-text-field type="text" label="Nome:"  
              v-model="usuario.nome" :error-messages="erros.nome" />  
          </v-flex>  
        </v-layout>  
      </v-container>  
    </v-content>  
  </v-app>  
</body>  
...
```

Implementando a API (inclusão)

■ usuarios-editar.html

...

```
<v-flex xs12>
  <v-text-field type="text" label="E-Mail:"
    v-model="usuario.email" :error-messages="erros.email" />
</v-flex>
<v-flex xs12>
  <v-text-field type="text" v-model="usuario.data"
    label="Data de Nascimento:" :error-messages="erros.data" />
</v-flex>
<v-flex xs12>
  <v-text-field type="password" label="Senha:"
    v-model="usuario.senha" :error-messages="erros.senha" />
</v-flex>
</v-layout>
<v-btn v-if="!modoEdicao()" color="primary" @click="inserir">
  <v-icon>save</v-icon> Inserir
</v-btn>
</v-container>
</v-content>
<v-footer dark app color="primary" class="pa-2">CORP&copy;</v-footer>
</v-app>
<script src="js/usuarios-service.js"></script>
<script src="js/usuarios-editar.js"></script>
</body>
</html>
```

Implementando a API (inclusão)

■ usuarios-listar.html

```
...  
    <v-btn flat icon href="usuarios-editar.html">  
        <v-icon>add</v-icon>  
    </v-btn>  
...
```

■ /js/usuarios-service.js

```
class UsuariosService {  
  
...  
  
    seleccionar(id) {  
        return this.request('get', '/' + id);  
    }  
  
...  
}
```

Implementando a API (inclusão)

■ /js/usuarios-editar.js

```
var router = new VueRouter({ mode: 'history', routes: [] });
```

```
var app = new Vue({  
  router,  
  el: '#app',  
  data() {  
    return {  
      mensagem: {},  
      usuario: {  
        id: null,  
        cpf: null,  
        nome: null,  
        email: null,  
        data: null,  
        senha: null,  
      },  
      erros: {  
        cpf: null,  
        nome: null,  
        email: null,  
        data: null,  
        senha: null,  
      }  
    }  
  },  
  ...  
});
```


Implementando a API (inclusão)

■ /js/usuarios-editar.js

```
...
methods: {
  modoEdicao(){
    return this.usuario.id ? true : false;
  },
  inserir() {
    this.limparMensagem();
    usuariosService.inserir(this.usuario)
      .then(id => {
        this.usuario.id = id;
        this.exibirMensagem('success',
          'Usuário com id = ' + id + ' criado com sucesso.');
```

...
 },
 tratarErro(error) {
 switch (error.response.status) {
 default:
 this.exibirMensagem('error', 'Erro inesperado.');

...
 },
 ...

Implementando a API (inclusão)

■ /js/usuarios-editar.js

```
...
    exibirMensagem(tipo, texto) {
        this.mensagem = { tipo, texto, exibir: true };
    },

    limparMensagem() {
        this.mensagem = { tipo: 'info', texto: '', exibir: false };
    },

}
});
```

Implementando a API (inclusão)

Curso RWS - Cadastro de x +

localhost:8080/cursorws/usuarios-editar.html

Pesquisar

< Curso RWS - Cadastro de Usuários


CPF:
44444444444

Nome:
Teste

E-Mail:
teste@gmailcom

Data de Nascimento:
01/01/2000

Senha:
.....

 INSERIR

CORP©

Implementando a API (atualização)

■ UsuariosBC.java

```
public class UsuarioBC {  
    public void atualizar(Usuario usuario) throws UsuarioNaoEncontradoException {  
        if (!repositorio.atualizar(usuario)) {  
            throw new UsuarioNaoEncontradoException();  
        }  
    }  
}
```

■ UsuariosRS.java

```
public class UsuariosRS {  
    @PUT  
    @Path("/{id}")  
    @Consumes(MediaType.APPLICATION_JSON)  
    @Produces(MediaType.APPLICATION_JSON)  
    public Response atualizar(@PathParam("id") Long id, Usuario usuario) {  
        try {  
            usuario.setId(id);  
            usuarioBC.atualizar(usuario);  
            return Response.ok(id).build();  
        } catch (UsuarioNaoEncontradoException e) {  
            throw new NotFoundException();  
        }  
    }  
}
```

Implementando a API (atualização)

■ usuarios-listar.html

```
...  
    <template slot="items" slot-scope="usuario">  
        <td>  
            <v-btn flat fab small  
                @click="editar(usuario.item.id)">  
                <v-icon>edit</v-icon>  
            </v-btn>  
            {{ usuario.item.cpf }}  
        </td>  
        ...  
    </template>  
...
```

■ usuarios-editar.html

```
...  
    <v-btn v-if="modoEdicao()" color="success" @click="atualizar">  
        <v-icon>save</v-icon> Atualizar  
    </v-btn>  
...
```

Implementando a API (atualização)

■ /js/usuarios-service.js

```
class UsuariosService {  
  ...  
  atualizar(usuario) {  
    return this.request('put', '/' + usuario.id, usuario);  
  }  
  ...  
}
```

Implementando a API (atualização)

■ /js/usuarios-editar.js

```
data() {...},

mounted() {
  this.usuario.id = this.$route.query.id;
  this.carregar();
},

methods: {
  ...
  carregar() {
    if (this.modaEdicao()) {
      usuariosService.selecionar(this.usuario.id)
        .then(usuario => {
          console.log(usuario);
          this.usuario = usuario;
        })
        .catch(error => {
          this.tratarErro(error);
        });
    }
  },
  ...
}
```

Implementando a API (atualização)

■ /js/usuarios-editar.js

```
...
  atualizar() {
    this.limparMensagem();
    usuariosService.atualizar(this.usuario)
      .then(data => {
        this.exibirMensagem('success', 'Usuário atualizado com sucesso.');
```

```
      })
      .catch(error => {
        this.tratarErro(error);
      });
  },

  tratarErro(error) {
    ...
    case 404: // Not found
      this.exibirMensagem('error',
        '0 registro solicitado não foi encontrado!');
      break;
    ...
  }
},
...

```





Implementando a API (atualização)

Curso RWS - Listagem de x +

localhost:8080/cursorws/usuarios-listar.html

Pesquisar

< Curso RWS - Listagem de Usuários

CPF ↑	Nome	E-mail	Data Nascimento
 11111111111	Pedro de Alcantara	pedro.alcantara@gmail.com	12/10/1798
 22222222222	Santos Dumont	santos.dumont@gmail.com	20/07/1873
 33333333333	Isabel de Braganca	maria@gmail.com	29/07/1846

Rows per page: 5 1-3 of 3 < > ↺ +

CORP©

Implementando a API (atualização)

Curso RWS - Cadastro de X +

localhost:8080/cursorws/usuarios-editar.html?id=1

Pesquisar

< Curso RWS - Cadastro de Usuários


CPF:
11111111111

Nome:
Pedro de Alcantara

E-Mail:
pedro.alcantara@gmail.com

Data de Nascimento:
12/10/1798

Senha:
.....

 ATUALIZAR

CORP©

Implementando a API (exclusão)

■ UsuariosBC.java

```
public class UsuarioBC {  
    ...  
    public Usuario excluir(Long id) throws UsuarioNaoEncontradoException {  
        Usuario usuario = repositorio.excluir(Usuario.class, id);  
        if (usuario == null) { throw new UsuarioNaoEncontradoException(); }  
        return usuario;  
    }  
}
```

■ UsuariosRS.java

```
public class UsuariosRS {  
    ...  
    @DELETE  
    @Path("{id}")  
    @Produces(MediaType.APPLICATION_JSON)  
    public Response excluir(@PathParam("id") Long id) {  
        try {  
            Usuario usuario = usuarioBC.excluir(id);  
            return Response.ok(usuario).build();  
        } catch (UsuarioNaoEncontradoException e) {  
            throw new NotFoundException();  
        }  
    }  
}
```

Implementando a API (exclusão)

■ /js/usuarios-service.js

```
class UsuariosService {  
  ...  
  excluir(id) {  
    return this.request('delete', '/' + id);  
  }  
  ...  
}
```

■ /js/usuarios-editar.js

```
...  
excluir() {  
  this.limparMensagem();  
  usuariosService.excluir(this.usuario.id)  
    .then(data => {  
    this.usuario = {id: null, cpf: null, nome: null,  
                    email: null, data: null, senha: null};  
    this.exibirMensagem('success', 'Usuário excluído com sucesso.');  })  
  .catch(error => {  
    this.tratarErro(error);  
  });  
},  
...  
}
```

Implementando a API (exclusão)

■ usuarios-editar.html

...

```
<v-btn v-if="modoEdicao()" color="error" @click="excluir">  
  <v-icon>delete</v-icon> Excluir  
</v-btn>
```

...

Implementando a API (exclusão)

Curso RWS - Cadastro de X +

localhost:8080/cursorws/usuarios-editar.html?id=1

Pesquisar

< Curso RWS - Cadastro de Usuários



CPF:
11111111111

Nome:
Pedro de Alcantara

E-Mail:
pedro.alcantara@gmail.com

Data de Nascimento:
12/10/1798

Senha:
.....

 EXCLUIR  ATUALIZAR

CORP©

Implementando a API (validação)

- Beans Validations

Implementando a API (validação)

■ Usuario.java

```
public class Usuario extends BaseModel implements Serializable {  
    @NotNull  
    @Size(min = 11, max = 11)  
    private String cpf;  
  
    @NotNull  
    @Size(min = 3, max = 100)  
    private String nome;  
  
    @NotNull  
    @Size(min = 1, max = 300)  
    private String email;  
  
    @NotNull  
    @Size(min = 6, max = 10)  
    private String senha;  
  
    @Past  
    @XmlJavaTypeAdapter(DateAdapter.class)  
    private Date data;  
    ...  
}
```


Implementando a API (validação)

■ UsuariosBC.java

@ApplicationScoped

```
public class UsuarioBC {
```

```
...
```

```
    public Long inserir(Usuario usuario) throws ValidacaoException {  
        validar(usuario);  
        return repositorio.inserir(usuario);  
    }
```

```
    public void atualizar(Usuario usuario) throws  
        UsuarioNaoEncontradoException, ValidacaoException {  
        validar(usuario);  
        if (!repositorio.atualizar(usuario)) {  
            throw new UsuarioNaoEncontradoException();  
        }  
    }
```

```
...
```

```
}
```

Implementando a API (validação)

■ UsuariosBC.java

```
private void validar(Usuario usuario) throws ValidacaoException {  
    Validator validator = Validation  
        .buildDefaultValidatorFactory().getValidator();  
    Set<ConstraintViolation<Usuario>> violations =  
        validator.validate(usuario);  
    if (!violations.isEmpty()) {  
        ValidacaoException validacaoException =  
            new ValidacaoException();  
        for (ConstraintViolation<Usuario> violation : violations) {  
            String entidade = violation  
                .getRootBeanClass().getSimpleName();  
            String propriedade = violation  
                .getPropertyPath().toString();  
            String mensagem = violation.getMessage();  
  
            validacaoException.adicionar(  
                entidade, propriedade, mensagem);  
        }  
        throw validacaoException;  
    }  
}
```

Implementando a API (validação)

■ UsuariosRS.java

```
@Path("usuarios")
public class UsuariosRS {
    ...
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Response inserir(Usuario body) {
        try {
            Long id;
            id = usuarioBC.inserir(body);
            String url = "/api/usuarios/" + id;
            return Response.status(Status.CREATED)
                           .header("Location", url)
                           .entity(id)
                           .build();
        } catch (ValidacaoException e) {
            return tratarValidacaoException(e);
        }
    }
    ...
}
```

Implementando a API (validação)

■ UsuariosRS.java

```
...
@PUT
@Path("/{id}")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public Response atualizar(@PathParam("id") Long id,
    Usuario usuario) {
    try {
        usuario.setId(id);
        usuarioBC.atualizar(usuario);
        return Response.status(Status.OK)
            .entity(id)
            .build();
    } catch (UsuarioNaoEncontradoException e) {
        throw new NotFoundException();
    } catch (ValidacaoException e) {
        return tratarValidacaoException(e);
    }
}
...
```

Implementando a API (validação)

■ UsuariosRS.java

...

```
private Response tratarValidacaoException(ValidacaoException e) {  
    return Response.status(Status.NOT_ACCEPTABLE)  
        .entity(e.getErros())  
        .build();  
}  
  
}
```

Implementando a API (validação)

■ usuarios-editar.js

```
$(function() {  
    ...  
    $("#salvar").click(function(event) {  
        limparMensagensErro();  
    });  
});  
  
function limparMensagensErro() {  
    /* Limpa as mensagens de erro */  
    $("#global-message").removeClass("alert-danger alert-success")  
        .empty().hide();  
    $(".control-label").parent().removeClass("has-success");  
    $(".text-danger").parent().removeClass("has-error");  
    $(".text-danger").hide();  
}  
    ...
```

Implementando a API (validação)

■ usuarios-editar.js

```
function tratarErro(request) {  
  switch (request.status) {  
    case 406:  
      $("form input").each(function() {  
        var id = $(this).attr("id"); var message = null;  
        $.each(request.responseJSON, function(index, value) {  
          if (id == value.propriedade) { message = value.mensagem; }  
        });  
        if (message) {  
          $("#" + id).parent().addClass("has-error");  
          $("#" + id + "-message").html(message).show();  
          $(this).focus();  
        } else {  
          $("#" + id).parent().removeClass("has-error");  
          $("#" + id + "-message").hide();  
        }  
      });  
      $("#global-message").addClass("alert-danger")  
        .text("Verifique erros no formulário!").show();  
      break;  
    }  
  }  
}
```

...

Implementando a API (validação)

Curso RWS x Fábio

localhost:8080/rs/usuarios-editar.html

Cadastro de Usuários

Verifique erros no formulário!

CPF

99999999999

tamanho deve estar entre 11 e 11

Nome

Nome completo

tamanho deve estar entre 3 e 100

Data de nascimento

99/99/9999

Email

nome@provedor.com

tamanho deve estar entre 1 e 300

Senha

tamanho deve estar entre 6 e 10

- Modelo de segurança comum:
 - Cookie no cliente + Session no servidor.
 - Uso de memória do servidor compromete escalabilidade.
 - Pouca informação é enviada ao cliente.
- Como garantir a segurança neste tipo de aplicação?
 - JOSE: JSON Object Signing and Encryption

- JOSE: JSON Object Signing and Encryption
 - Padrão que fornece uma abordagem geral para a assinatura e criptografia de qualquer conteúdo, não necessariamente no JSON.
 - No entanto, é construído utilizando JSON e BASE64URL para ser facilmente utilizável em aplicações WEB.
 - BASE64URL
 - Base64: Algoritmo para representar bytes em ([A-Z],[a-z],[0-9], "/", "+" e "=")
 - URL: Substitui "/" por "_", "+" por "-" e "=" remove

- JOSE: Consiste em vários RFC's:
 - JWA: JSON Web Algorithms, descreve algoritmos de criptografia.
 - JWK: JSON Web Key, descreve o formato e o tratamento de chaves de criptografia.
 - JWS: JSON Web Signature, descreve a produção e o tratamento de mensagens assinadas.
 - JWE: JSON Web Encryption, descreve a produção e o tratamento de mensagens criptografadas.
 - JWT: JSON Web Token, descreve a representação de requisições codificadas em JSON e protegidas por JWS ou JWE.

■ JWT: JSON Web Token

- É simplesmente o hash do JSON com reivindicações (claims), que é assinado com JWS ou criptografado com JWE e serializado de forma compacta.
- Formado por 3 partes separadas por pontos (.):
 - Header: Cabeçalho
 - Payload: Dados que serão distribuídos
 - Signature: Assinatura de “header.payload”
- Define reivindicações (claims) padrão.
- Podemos adicionar novos.

■ JWT: JSON Web Token

■ Header

- Consiste em duas partes codificadas em Base64URL:
 - O tipo (JWT)
 - O algoritmo de hash usado (HMAC SHA256 ou RSA).
 - Ex: {“alg” : “HS256”, “typ” : “JWT”}
- Então teríamos a primeira parte
 - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
 -
 - yyyyyy
 -
 - zzzzzz

■ JWT: JSON Web Token

■ Payload

- Contém a informação (claims) em si compartilhada entre as partes. Basicamente, é o dado de interesse transmitido.
- Também codificado em Base64:
 - {"nome" : "Fulano", "admin" : true}
- Então teremos:
 - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
 -
 - eyJub21lIjoIbnVsYW5vliwiYWRTaW4iOnRydWV9
 -
 - zzzzzzz

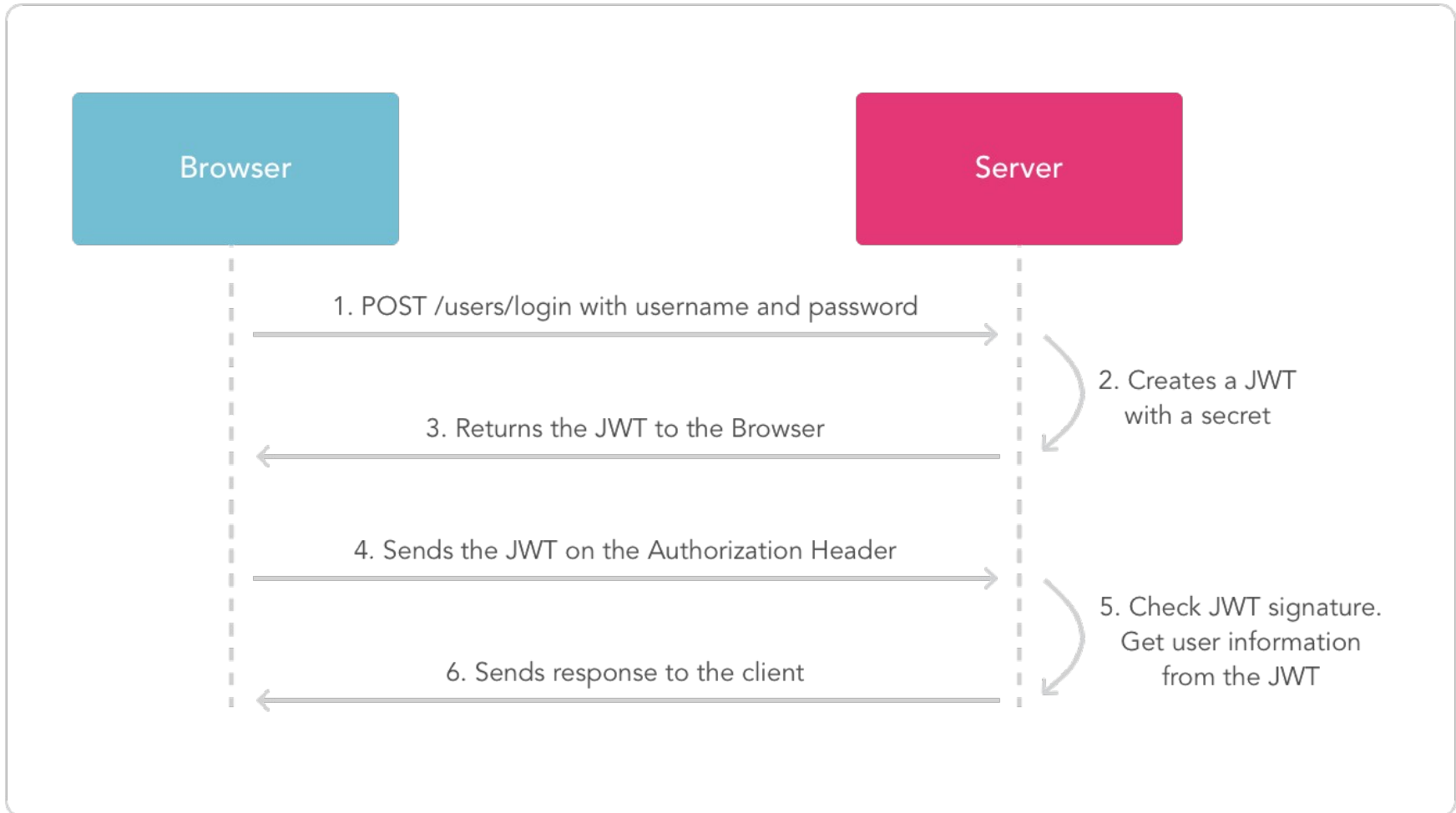
- JWT: JSON Web Token
 - Reivindicações padrão:
 - "iss" identifica o emitente da reivindicação
 - "sub" identifica o assunto da JWT
 - "aud" (audiência) identifica os destinatários
 - "exp" marca o tempo de expiração do JWT
 - "nbf" (não antes) marca o tempo antes do qual JWT deve ser rejeitado
 - "iat" (emitido em) marca o tempo quando JWT foi criado
 - "jti" (JWT ID) identificador exclusivo para JWT

■ JWT: JSON Web Token

■ Signature

- Para assinar o token, usamos o Header, Payload, o algoritmo definido no header, uma chave.
- Exemplo usando o HMAC SHA256 (HS256):
 - $\text{HMACSHA256}(\text{header} + "." + \text{payload}, \text{secret})$
- Onde header e payload já estão codificados em Base64.
- Finalmente:
 - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
 -
 - eyJub21lljoiRnVsYW5vliwiYWRTaW4iOnRydWV9
 -
 - IShPdPgMqjygLcv6FpePbFuRLJHBTdeKSNDQIpR-X2E

■ JWT: Como funciona?



- JWT: Como funciona?
 - Quando o usuário faz o login com sucesso, o servidor retorna o JWT que deve ser salvo localmente.
 - cookie, localStorage, sessionStorage, etc.
 - A cada requisição, envia-se o token recebido.
 - O servidor recebe e, dado que ele possui chave e o algoritmo que está no header, consegue validar a informação do payload e usá-la sem precisar ir ao banco de dados todas as vezes.
 - O servidor gera um novo token com nova data de expiração e retorna pra o cliente.

- JOSE: JSON Object Signing and Encryption
 - <https://datatracker.ietf.org/wg/jose/charter/>
- JWT: JSON Web Tokens
 - <https://tools.ietf.org/html/draft-ietf-oauth-json-web-token>
- JWS: JSON Web Signing
 - <http://tools.ietf.org/html/draft-ietf-jose-json-web-signature>
- JWE: JSON Web Encryption
 - <http://tools.ietf.org/html/draft-ietf-jose-json-web-encryption>
- JWT Authorization Grants
 - <http://tools.ietf.org/html/draft-ietf-oauth-jwt-bearer>
- JWK: JSON Web Keys
 - <http://tools.ietf.org/html/draft-ietf-jose-json-web-key>