

**Aula 1****FRAMEWORK JAVA EE**

*Visão geral dos principais componentes, do front até o back-end: JSPs, Servlets, Session e Entity Beans.*

49 minutos

**INTRODUÇÃO**

Caro estudante, para iniciarmos este estudo, é importante que você saiba que Java é uma das linguagens de programação mais utilizadas no mundo. Ela surgiu no ano de 1995 e utiliza o paradigma da Programação Orientada a Objetos para o desenvolvimento de novas aplicações. Ao longo do tempo, foram estruturadas edições para uso e escolha pelos desenvolvedores: Java Standard Edition (Java SE); Java Enterprise Edition (Java EE); e Java Micro Edition (Java ME).

Nesta aula, você entenderá mais sobre o Java Enterprise Edition, destinado à utilização da linguagem Java em ambientes empresariais que requerem computação distribuída, uso otimizado de recursos e escalabilidade, conforme crescimento no uso da aplicação desenvolvida.

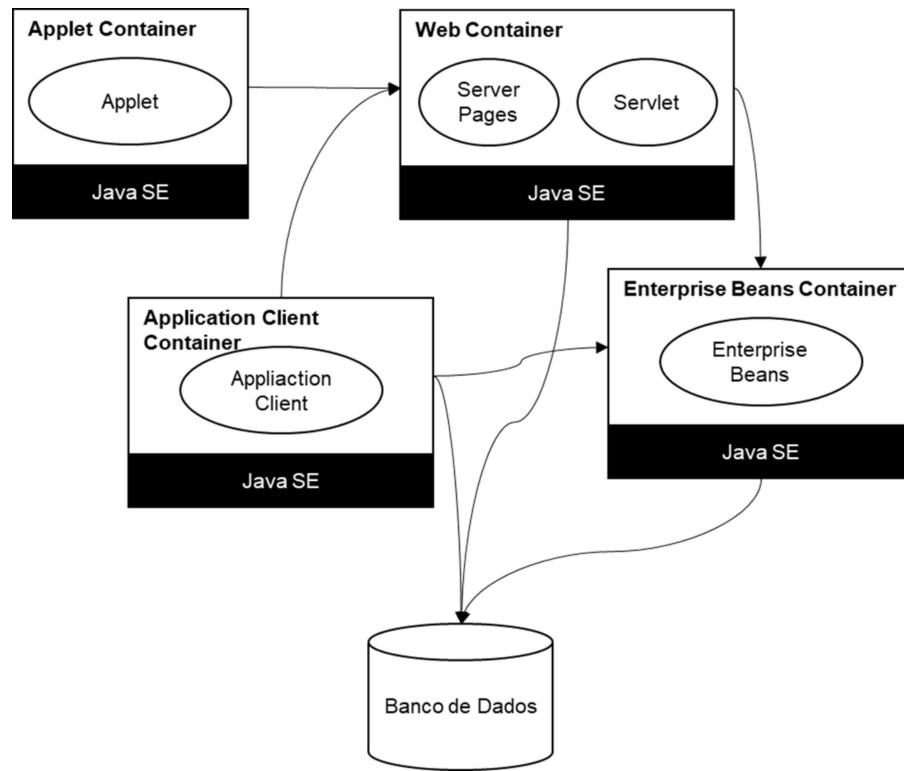
Você está pronto para entender como construir aplicações usando o Java EE? Espero que sim. Bons estudos!

**ARQUITETURA JAVA EE**

Neste bloco, você aprenderá sobre o framework Java EE ou, ainda, Jakarta EE. E, antes de começarmos, vale a pena um breve histórico. As primeiras versões desse framework usavam a nomenclatura J2EE, que foi bastante comum de 1999 a 2006. A partir de então, passou-se a usar a nomenclatura Java EE até 2019, quando o nome Jakarta EE passa a ser oficialmente utilizado (TIJMS, 2020). Importante também que você saiba que a Sun, empresa originalmente criadora do Java (e do Java EE) foi adquirida pela Oracle em 2009.

Jakarta EE Platform se trata de uma plataforma para hospedar aplicações Jakarta EE de acordo com Jakarta EE Specification 9.1 (SPECIFICATIONS, [s. d.]). A plataforma Jakarta EE contempla uma série de elementos criados e definidos para possibilitar a elaboração de aplicações que devem seguir as especificações e orientações da plataforma. Na Figura 1, você poderá visualizar os principais macrocomponentes dela.

Figura 1 | Arquitetura plataforma Jakarta EE



Fonte: adaptada de Jakarta EE Platform 9.1 (2021).

Nessa figura você pode observar quatro elementos denominados como *container* e que são: *Applet Container*, *Application Client Container*, *Web Container* e *Enterprise Beans Container*. Cada *container* apresenta um conjunto de recursos para o desenvolvimento de uma aplicação específica. Repare, ainda, que cada um deles é baseado no Java SE (ou Java Standard Edition), que é a linguagem de programação Java e que, portanto, as aplicações escritas em cada um seguem também as regras e especificações da linguagem Java.

As aplicações clientes (ou *Application Clients*) são programas escritos em Java (geralmente com interfaces gráficas), os quais executam no computador do próprio usuário. Já os *Applets* são componentes gráficos que, em geral, são executados no navegador web. O *Web Container* permite a execução de *servlets*, *server pages*, *server faces applications*, *filters* e *web event listeners*, os quais podem ser usados para gerar páginas HTML a serem usadas para a interface da aplicação com um usuário. Em geral, serão executadas em um conjunto de servidores que atenderão a uma série de requisições originadas por um navegador web. Para cada requisição se devolverá o conteúdo necessário a ser consumido e exibido por um navegador. Por fim, os *Enterprise Beans* correspondem a componentes que contêm lógica de negócios para uma aplicação e que são executados em um ambiente gerenciado que requer suporte a transações.

Todos esses *containers* poderão também se comunicar com o banco de dados para executar transações específicas, como gravar, alterar ou apagar dados em tabelas. Além disso, poderão realizar consultas a esses dados. A plataforma, contudo, estimula que essas comunicações aconteçam de forma organizada e por camadas, possibilitando assim o uso do padrão MVC (*Model-View-Controller*).

Com isso, foi possível você conhecer os principais elementos que fazem parte da plataforma Jakarta EE. Além disso, é relevante pontuar também que, para cada um dos *containers* mencionados, existe um conjunto de funcionalidades para o desenvolvimento de recursos que serão executados dentro do seu ambiente.

Mais um ponto importante: a especificação da plataforma Jakarta EE define o conjunto de regras, diretrizes e políticas que governarão tudo o que for criado ou desenvolvido. Diferentes fabricantes podem criar diferentes produtos da plataforma contanto que obedeçam ao que for definido na especificação. Assim, cada fabricante poderá oferecer recursos adicionais e complementares para a plataforma. Exemplos de fabricantes são: Oracle, IBM, Red Hat dentre outros. Há também a implementação de referência *open-source* denominada Eclipse Glassfish.

As implementações criadas pelos fabricantes com o intuito de suportar a especificação da plataforma Jakarta EE ou Java EE são chamadas, em geral, de servidores de aplicação. Os *containers* chamados *Web Container* e *Enterprise Beans Container* residem no servidor de aplicação e permitem a execução do código elaborado por um desenvolvedor.

Nesse sentido, é importante que você conheça mais sobre esses dois *containers* que são amplamente utilizados no mercado para o desenvolvimento de diversas aplicações web. É isso que faremos nos próximos blocos.

Bons estudos!

#### **VIDEOAULA: ARQUITETURA JAVA EE**

Você conheceu a arquitetura da plataforma Jakarta EE ou Java EE. Neste vídeo, você aprenderá mais alguns detalhes sobre a plataforma, os quais contemplam: ciclo de vida, gerenciamento de transações e segurança. Também verá como se dá o fluxo de informações entre a aplicação web e os enterprise beans.

Bons estudos!

Para visualizar o objeto, acesse seu material digital.

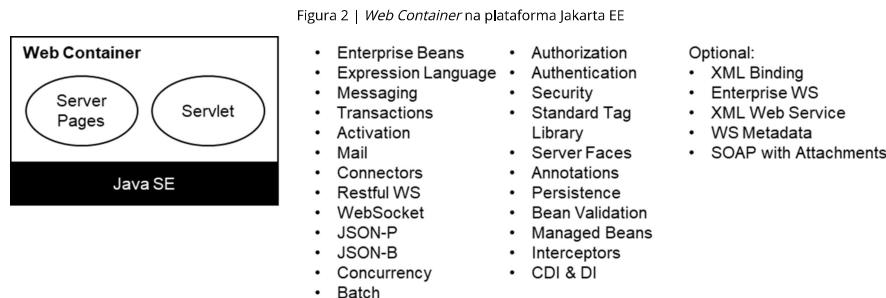
#### **DESENVOLVIMENTO WEB**

##### **Introdução**

O *Web Container* permite a construção de interfaces voltadas para o ambiente web e que são executadas em um servidor de aplicação Jakarta EE. Quando um usuário solicita, no navegador web, uma página, ela será processada e devolvida pelo *Web Container* nesse servidor de aplicação. Essa página poderá ser resultado da execução de um *servlet*, de um *Java Server Pages* (JSP) ou de um *server faces*.

##### **Web Container**

A Figura 2 apresenta uma visualização das diversas funcionalidades existentes dentro do *Web Container*. Essas funcionalidades são implementadas e oferecidas por um servidor de aplicação de tal forma que o desenvolvedor possa se preocupar com a entrega que ele precisa fazer, já que aspectos diversos, como tratamento de transações, conectividade com bancos de dados, autenticação, autorização, entre outros, são todos tratados e gerenciados pelo servidor de aplicação.



Fonte: adaptada de Jakarta EE Platform 9.1 (2021).

##### **Servlets**

De acordo com a especificação Jakarta Servlet... (2020), *servlet* é um componente web gerenciado pelo *container* do servidor de aplicação e que gera conteúdo dinâmico. *Servlets* são classes Java independentes de plataforma que são compiladas e carregadas dinamicamente pelo servidor de aplicação para atender a requisições externas.

Com a finalidade de construir *servlets*, definições são realizadas em arquivos chamados de *Deployment Descriptors* ou mesmo através do recurso de *Annotations* da linguagem Java. Dessa forma, uma chamada, via navegador, a uma URL será associada a um *servlet* e, de acordo com os métodos chamados (GET, POST, UPDATE, DELETE), a implementação existente será, então, executada.

Quadro 1 | Exemplo de código fonte para *servlet*

```

1  public class CalculatorServlet extends HttpServlet {
2      public void doGet(HttpServletRequest req, HttpServletResponse res)
3      {
4          ...
5          ...
6      }
7  }

```

Fonte: adaptado de Jakarta Servlet... (2020).

Um aspecto importante em relação aos *servlets* é que o próprio código deve gerar o conteúdo HTML a ser devolvido para o navegador através do uso de funcionalidades presentes na plataforma.

### **Java Server Pages (JSP)**

Para melhor separar o chamado código de apresentação da lógica de negócios, a plataforma introduziu o recurso chamado *Java Server Pages* ou simplesmente JSP. Nesse componente, os códigos HTML estão presentes em todo o corpo do documento, e referências a variáveis ou conteúdos gerenciados por objetos são referenciados quando necessário para a construção da página.

No Quadro 2, extraído da própria especificação JSP, tem-se um código para você compreender como esse mecanismo funciona. O exemplo cria uma página HTML com o intuito de exibir uma lista de filmes.

Quadro 2 | Exemplo de código fonte JSP

```

1  <html xmlns="http://www.w3.org/1999/xhtml"
2      xmlns:jsp="http://java.sun.com/JSP/Page"
3      xmlns:c="http://java.sun.com/jsp/jstl/core"
4      xmlns:u="urn:jsptagdir:/WEB-INF/tags/mylib/"
5      xmlns:data="http://acme.com/functions">
6      <c:set var="title" value="Best Movies" />
7      <u:headInfo title="${title}" />
8      <body>
9          <h1>${title}</h1>
10         <h2>List of Best Movies</h2>
11         <ul>
12             <c:forEach var="m" varStatus="s" items="data:movieItems()">
13                 <li><a href="#EL${s.index}">${s.index}</a>${m.title}</li>
14             </c:forEach>
15         </ul>
16     </body>
17 </html>

```

Fonte: Jakarta Server Pages [s.p] (2020).

Na linha 07, há o uso de **u:headInfo** que, após realizar o processamento, gerará o conteúdo seguinte. Na linguagem JSP, esse recurso é chamado de **custom action** e são *tags* que internamente executam um código que produzirá uma saída esperada. Note que, na linha 06, definiu-se o valor em uma variável, a qual foi usada para construir o resultado a seguir:

```
<head><title>Best Movies</title></head>
```

O recurso **custom action** é bastante usado para construir páginas JSP e faz parte do que é chamado *tag libraries*. Na linha 12, pode-se verificar o uso desse recurso para construir um loop, que exibirá uma lista de filmes a partir de uma chamada a um método chamado movieItems() de um objeto chamado **data**. Existem várias *tag libraries* prontas para uso na especificação da linguagem e o próprio desenvolvedor pode criar *tags* adicionais.

### **Server faces**

Apesar de existirem vários recursos bastante poderosos tanto para *servlets* quanto para JSPs, ainda havia alguns desafios com relação ao uso dessas tecnologias, os quais envolviam o gerenciamento de fluxos de controle, ou *workflow*, entre as páginas, naveabilidade entre páginas e tratamento de erros envolvendo esse

processamento. Dessa forma, criou-se uma especificação para tornar mais simples a construção de interfaces gráficas, melhorar o gerenciamento de estado entre as páginas e surgiu, então, o recurso *Jakarta Server Faces*.

Esse recurso possibilita uma divisão mais clara entre desenvolvedores de código, autores de páginas, desenvolvedores de componentes reusáveis e provedores de ferramentas visuais para auxiliar o trabalho de programação. Há várias diferenças em relação a como ocorre o processamento de *server faces* em relação às *server pages*.

O Quadro 3 mostra um código construído com *server faces*. Você poderá notar certa semelhança com o código JSP, mas verá que a estrutura e outros recursos para controle e gerenciamento do fluxo das páginas são bastante diferentes. Assim, pode-se dizer que, de fato, a tecnologia *server faces* foi uma evolução natural do JSP.

Quadro 3 | Exemplo de código fonte server faces

```

1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3  <html lang="en"
4      xmlns="http://www.w3.org/1999/xhtml"
5      xmlns:h="http://xmlns.jcp.org/jsf/html">
6      <h:head>
7          <title>Facelets Hello Greeting</title>
8      </h:head>
9      <h:body>
10         <h:form>
11             <h:graphicImage
12                 url="#{resource['images:duke.waving.gif']}"
13                 alt="Duke waving his hand"
14             />
15             <h2>Hello, my name is Duke. What's yours?</h2>
16             <h:inputText id="username"
17                 title="My name is: "
18                 value="#{hello.name}"
19                 required="true"
20                 requiredMessage="Error: A name is required."
21                 maxlength="25" />
22             <p></p>
23             <h:commandButton
24                 id="submit"
25                 value="Submit"
26                 action="response">
27             </h:commandButton>
28             <h:commandButton
29                 id="reset"
30                 value="Reset"
31                 type="reset">
32             </h:commandButton>
33         </h:form>
34         ...
35     </h:body>
36 </html>
```

Fonte: The Jakarta [s.p]. (2020).

A partir disso, você pode compreender *server faces* como um framework MVC para construir interfaces para os usuários em aplicações web incluindo componentes de interface com usuário, gerenciamento de estado, manipulação de eventos, validação de entrada de dados, navegação de páginas e suporte para internacionalização e acessibilidade.

### Conclusões

Neste bloco, você conheceu os principais recursos para o desenvolvimento *front-end* presentes na plataforma Jakarta EE (ou Java EE). *Servlets*, *server pages* e *server faces* são bastante utilizados.

Bons estudos!

## VIDEOAULA: DESENVOLVIMENTO WEB

Que tal entender mais sobre os códigos apresentados neste bloco e aprimorar seus conhecimentos quanto aos *servlets*, *JSPs* e *server faces*? Neste vídeo você aprenderá detalhes adicionais referentes a esses componentes.

Bons estudos!

Para visualizar o objeto, acesse seu material digital.

### **ENTERPRISE BEANS**

#### **Introdução**

Neste bloco, você estudará os *enterprise beans*, que são recursos da plataforma Jakarta EE que contêm lógica de negócios para realizar operações transacionais com bancos de dados. Na execução, os *enterprise beans* obedecem a um contexto transacional e são gerenciados pelo *container* no servidor de aplicação.

Existem dois tipos de *enterprise beans*: o primeiro diz respeito aos *session beans*, que são usados para a construção da lógica de negócios, e o segundo são os *entity beans*, os quais representam uma tabela do banco de dados e suas linhas. Os *entity beans* representam objetos de acesso aos dados e estão associados a um mapeamento objeto-relacional.

Na Figura 3, você pode visualizar os recursos que são oferecidos para *enterprise beans* por um servidor de aplicação para a plataforma Jakarta EE.

Figura 3 | Enterprise Beans Container na plataforma Jakarta EE



Fonte: adaptada de Jakarta EE Platform 9.1 (2021).

Os *enterprise beans* são instalados e configurados no servidor de aplicação e são catalogados em um serviço de localização chamado *Java Naming Directory Interface* (JNDI). Além deles, outros recursos, como conexões a banco de dados, filas JMS e outros, também são catalogados, criados e gerenciados pelo servidor de aplicação.

#### **Enterprise beans**

Existem alguns tipos de *enterprise beans* na plataforma Jakarta EE: *session beans*, *message-driven beans* e *entity beans*. Todos esses recursos têm o ciclo de vida gerenciado pelo servidor de aplicação. Isso significa que ele cria um *pool* de recursos para atender às requisições e, assim, gerenciar o uso deles pelos clientes com o intuito de fazer melhor uso dos recursos computacionais.

#### **Session beans**

São componentes que executam lógica de negócios. Em geral são utilizados para trabalhar de forma transacional com os *entity beans* em um modelo chamado de *session façade*. Nesse sentido, podem realizar cálculos ou outras operações a partir de informações oriundas de um meio persistente, como banco de dados, e efetivá-las quando necessário.

No servidor de aplicações, os *session beans* são instanciados e ficam disponíveis para uso das aplicações. Para usar um deles é necessário solicitar a sua criação através da interface *Home*, permitindo, assim, que o servidor de aplicação faça toda a gestão do ciclo de vida, dos recursos computacionais (como memória e uso de processador), da segurança e do controle transacional.

*Session beans* podem ser *stateful* ou *stateless*. Um *stateless session bean* é um recurso compartilhado por vários clientes e que não mantém nenhuma informação de estado entre as chamadas. Já os *stateful session beans* são disponibilizados para um único cliente, e o estado é mantido durante as diversas chamadas.

efetuadas pelo cliente. Veja exemplos de declaração nos Quadros 4 e 5. No Quadro 5, você pode ver também um exemplo de chamada ao *session bean*.

Quadro 4 | Declaração de um *stateless session bean*

```

1  Stateless
2  public class MyStatelessEjb {
3      public String sayHello(String name) {
4          return "Olá, " + name + "!";
5      }
6  }
```

Fonte: elaborado pelo autor.

Quadro 5 | Declaração de um *stateful session bean*

```

1  @Stateful
2  public class MyStatefulEjb {
3  }
```

Fonte: elaborado pelo autor.

Quadro 6 | Exemplo de chamada de um *stateless session bean*

```

1  public class MinhaClasse {
2      @EJB
3      MyStatelessEjb ejb;
4      public void test() {
5          ejb.sayHello("Fulano");
6      }
7  }
```

Fonte: elaborado pelo autor.

### **Jakarta Persistence API**

Trata-se de uma API para o gerenciamento de persistência e de mapeamento objeto-relacional que pode ser usado tanto em ambientes de projeto Java SE quanto nos de Jakarta EE. Objetos Java são usados para serem mapeados com tabelas de banco de dados, as quais são chamadas de classes de entidade.

No Quadro 7, você pode ver um código-fonte referente à declaração de uma entidade chamada *Customer*. A menos que indicado o contrário, essa entidade estará associada a uma tabela CUSTOMER. Da mesma forma, atributos da classe são associados a colunas na tabela no banco de dados.

Quadro 7 | Declaração de uma entidade

```

1  @Entity
2  public class Customer implements Serializable {
3      private Long id;
4      private String name;
5      // No-arg constructor
6      public Customer() {}
7      @Id // property access is used
8      public Long getId() {
9          return id;
10     }
11     public void setId(Long id) {
12         this.id = id;
13     }
14     public String getName() {
15         return name;
16     }
17     public void setName(String name) {
18         this.name = name;
19     }
20 }

```

Fonte: elaborado pelo autor.

### Conclusão

A plataforma Jakarta EE oferece uma série de recursos tecnológicos para a construção de aplicações distribuídas. Os *enterprise beans* são elementos que permitem a disponibilização rápida e eficiente de lógica de negócios para a camada de *front-end*.

### VIDEOAULA: ENTERPRISE BEANS

Você aprendeu neste bloco algumas informações importantes para conhecer os *enterprise beans*. Nesta videoaula, o objetivo é que você conheça também os *session beans* do tipo *singleton*, *stateful* e *message-driven*, além de complementar o conhecimento quanto à *Persistence API*.

Bons estudos!

Para visualizar o objeto, acesse seu material digital.

### ESTUDO DE CASO

Para contextualizar a sua aprendizagem, imagine que você seja o arquiteto técnico em uma equipe de desenvolvimento com foco na linguagem Java e que precisa criar um sistema para a gestão de frotas de veículos da empresa em que trabalha.

Dentre as tecnologias apresentadas ao longo da aula, quais você selecionaria para a camada de apresentação ou *front-end*?

Quais você selecionaria para o *back-end*?

De forma complementar, como você poderia escolher o servidor de aplicação a utilizar?

### RESOLUÇÃO DO ESTUDO DE CASO

Para o estudo de caso apresentado, possivelmente você pode ter pensado em uma combinação das tecnologias apresentadas na aula. Com este vídeo, espera-se complementar seus pensamentos com novas reflexões para que você perceba como isso pode impactar na evolução do projeto e como isso pode permitir que você tenha novas perspectivas a respeito dessa escolha.

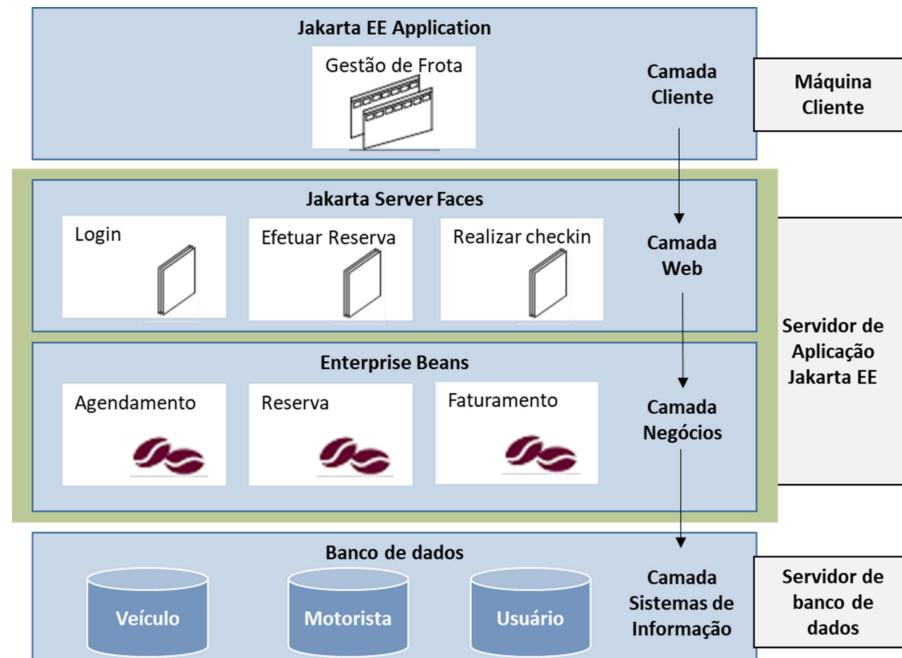
Tendo como exemplo um sistema de frotas, é possível pensar em algumas funcionalidades necessárias. Por exemplo, para a camada de persistência, entidades no banco de dados como Frota, Veículo, Motorista e Usuário são possibilidades reais para se ter num sistema como esse. Com isso, as funcionalidades *insert*, *update*, *delete* e *select* são necessárias para cada uma dessas entidades.

Já para a camada de lógica de negócios, algumas funcionalidades também poderão ser criadas, por exemplo: `agendarVeículo()`, `checkin()`, `checkout()`, `associarMotoristaAoVeículo()`, `realizarFaturamento()`.

Na camada de apresentação, existirão páginas que contemplarão essas funcionalidades mencionadas na lógica de negócios em seu aspecto visual. Montar páginas para autenticar e autorizar acesso, cadastrar veículos e motoristas e realizar os agendamentos.

Considerando os recursos disponíveis na plataforma Jakarta EE, a camada de apresentação pode ser construída com o recurso `server faces`. As funcionalidades de lógica de negócios estarão em `enterprise Java beans` e as entidades necessárias e associadas ao banco de dados serão mapeadas via `Jakarta Persistence API`. A Figura 4 contém um desenho representando a distribuição de alguns desses componentes, como componentes `Jakarta server pages` e `enterprise beans` rodando em um servidor de aplicação Jakarta EE. A comunicação entre a camada de negócios e a camada de sistemas de informação ocorre via `Jakarta Persistence API`.

Figura 4 | Sistema de frotas de veículos com Jakarta EE



Fonte: elaborada pelo autor.

Resolução do Estudo de Caso

Para visualizar o objeto, acesse seu material digital.

### 💡 Saiba mais

Para conhecer melhor os conceitos que envolvem frameworks, vale a pena conferir o vídeo a seguir:

[https://www.youtube.com/watch?v=2zqzzTnfa0E&ab\\_channel=C%C3%B3digofonteTV](https://www.youtube.com/watch?v=2zqzzTnfa0E&ab_channel=C%C3%B3digofonteTV)

E para conhecer mais sobre framework Spring e sua estrutura, acesse:

<https://spring.io/>

## Aula 2

# FRAMEWORKS JAVASCRIPT

*Visão geral dos frameworks baseados em JavaScript.*

*48 minutos*

### INTRODUÇÃO

Nesta aula você conhecerá alguns frameworks para o desenvolvimento com JavaScript, que é uma das linguagens de programação mais populares do mercado. Enquanto HTML especifica o conteúdo da página e CSS especifica o layout, ou desenho, da página, com JavaScript você pode programar o comportamento das páginas.

Existem vários frameworks para desenvolvimento JavaScript disponíveis no mercado. Nesta aula você terá contato com o Ajax, o Angular e o Node.js, que são bastante utilizados. Com isso, terá a possibilidade de aprimorar seu desenvolvimento em conjunto com outras plataformas.

Espera-se que, com esta aula, você consiga ter uma visão geral sobre esses frameworks e possa incrementar seu aprendizado buscando mais conhecimento sobre essas tecnologias.

Bons estudos!

### MUNDO JAVASCRIPT

Como JavaScript é uma linguagem de programação bastante utilizada no mercado, é imprescindível que você conheça os principais frameworks construídos para melhorar o processo de desenvolvimento no dia a dia. Isso significa que esses frameworks o apoiarão, com uma criação mais rápida, para dar conta das várias necessidades presentes na rotina do desenvolvimento, bem como para buscar uma padronização nas diretrizes de código e consistência quanto à forma de se trabalhar.

Além disso, os frameworks também ajudarão um código que atuará de uma maneira mais simplificada com o *Document Object Model* (DOM) das páginas HTML. Ele também ajudará a criar estruturas de código independentes do tipo de navegador ou versão utilizado.

### Ajax

O Ajax é usualmente considerado o acrônimo de *Asynchronous JavaScript And XML*. É uma tecnologia que executa no cliente, no próprio navegador, e que é usado para atualizar os conteúdos de uma página sem a necessidade de recarregá-la. Esse termo surgiu após a disponibilização, pela Microsoft em 2000-2002, de um objeto chamado XMLHttpRequest no navegador Internet Explorer para realizar a comunicação com um servidor.

Com a introdução desse objeto, foi possível interagir com o servidor sem a necessidade de recarregar a página. Assim, receber ou enviar dados passaram a ser tarefas feitas via JavaScript, sem a necessidade de submeter e receber uma nova página pelo navegador. No Quadro 1 se tem um exemplo de código que instancia um objeto XMLHttpRequest (linha 03), solicita um conteúdo remoto (linhas 09 e 10) e atualiza um conteúdo HTML chamado demo com o retorno (linha 06).

Quadro 1 | Exemplo de código com Ajax

```

1 <script language="Javascript">
2   function loadDoc() {
3     var xhttp = new XMLHttpRequest();
4     xhttp.onreadystatechange = function() {
5       if (this.readyState == 4 && this.status == 200) {
6         document.getElementById("demo").innerHTML = this.responseText;
7       }
8     };
9     xhttp.open("GET", "conteúdo_remoto.txt", true);
10    xhttp.send();
11  }
12 </script>

```

Fonte: W3SCHOOLS (c2022).

## Frameworks JavaScript

Existem diversos frameworks disponibilizados no mercado. Alguns têm uma comunidade de desenvolvedores bastante ativa, outros possuem um envolvimento maior de grandes empresas, além de frameworks com algum grau de risco, que podem prover de novas criações ou de outros que não tiveram boa adesão no mercado.

Se existem tantos frameworks no mercado, como escolher qual deles utilizar? A resposta a essa pergunta dependerá das necessidades da aplicação a ser desenvolvida, se os usuários interagirão mais com navegadores em computadores ou em dispositivos móveis, se haverá ou não entrada de dados, eventuais custos de licenciamento ou a própria experiência da equipe em usar ou aprender um framework.

Na Figura 1 você poderá ver alguns frameworks disponíveis para JavaScript de acordo com levantamento apontado por Ermigliotti (2021). Na lista, o autor classifica os frameworks em *front-end*, *back-end* e testes. Como frameworks para *front-end*, o autor lista **Angular**, Vue JS, Next JS, React JS, Ember JS, Svelte JS, Gatsby JS, Nuxt JS e Bootstrap. Como frameworks para *back-end*, estão presentes **Node.js**, Spring boot, Express JS, Laravel e Micronaut. O autor também apresenta alguns frameworks voltados para testes como Mocha.js, Jasmine e Jest.

Figura 1 | Frameworks JavaScript



Fonte: Ermigliotti [s. p.]. (2021).

Como você pode observar, existem vários frameworks e cada um deles possui uma ou mais aplicações recomendadas. Os frameworks JavaScript, em geral, dividem sua arquitetura entre o processamento *client-side* e *server-side*. Alguns frameworks também reusam ou possuem como pré-requisitos outros frameworks. É bastante comum encontrar frameworks JavaScript construídos a partir de Node.js por exemplo.

### Conclusão

Com esta aula você teve a oportunidade de conhecer Ajax e de ter uma visão sobre os frameworks para JavaScript considerando as camadas de *front-end* e *back-end*. Como sugestão, visite alguns sites dos projetos indicados para ter uma percepção melhor da utilização de cada um deles.

Bons estudos!

## VIDEOAULA: MUNDO JAVASCRIPT

Neste vídeo, vamos conversar sobre o surgimento da linguagem JavaScript como uma forma de introduzir comportamentos para o conteúdo exibido pelas páginas HTML em conjunto com CSS e como surgiu a necessidade de se construir um tipo de programação que se convencionou chamar de Ajax. Ao longo dos novos JavaScript, ele foi tendo cada vez maior importância e vários frameworks foram, então, criados.

Bons estudos!

Para visualizar o objeto, acesse seu material digital.

## NODE.JS

Neste bloco você terá contato com um framework JavaScript bastante conhecido e utilizado, que é o Node.js. Você poderá encontrar os binários para instalação no seguinte site: <https://nodejs.org/>. Além disso, lá encontrará guias, tutoriais e outras informações sobre o projeto.

De acordo com informações do próprio site do projeto, em Introduction... (2020), Node.js é um ambiente de execução JavaScript multiplataforma e *open-source*. É uma ferramenta popular para vários tipos de projetos e foi construído a partir da engine V8 JavaScript do Google Chrome.

Node.js é uma plataforma que facilita a comunicação assíncrona entre um cliente e o servidor e é, portanto, um ambiente de execução de código JavaScript. Apesar de facilitar muito o trabalho de desenvolvimento, diversos outros frameworks foram criados a partir do Node.js com diferentes especificidades.

Uma das vantagens apresentadas por Node.js é a possibilidade de escrever tanto o *front-end* quanto o *back-end* de uma aplicação com JavaScript. Ou seja, ter tanto o código no cliente quanto no servidor escritos em JavaScript, mas, mais do que isso, existe a possibilidade de evoluir o código JavaScript no servidor sem ficar preso às versões suportadas pelo navegador no cliente.

Por ter um servidor web rodando no servidor, todas as requisições enviadas serão tratadas por um único processo no sistema operacional, sem a necessidade de se criar uma *thread* de execução para atender cada uma delas. Node.js pode então tratar milhares de instruções concorrentes ao mesmo tempo em que realiza leituras da rede, acessa o banco de dados ou mesmo o sistema de arquivos.

Após você instalar o Node.js em seu computador, você pode criar um arquivo com o nome index.js e colocar o código do Quadro 2. Na Figura 2, o servidor Node.js é inicializado pela linha de comando e, na Figura 3, você pode visualizar o resultado obtido.

Quadro 2 | Código de exemplo em Node.js

```

1  var http = require('http')
2  const hostname = '127.0.0.1'
3  const port = 3000
4  const server = http.createServer((req, res) => {
5      var html = buildHtml(req);
6      res.writeHead(200, {'Content-Type': 'text/html'})
7      res.write(html)
8      res.end();
9  })
10 function buildHtml(req) {
11     var header = '<head><title>Oi</title></head>';
12     var body = '<h1>Oi, tudo bem?</h1><p>Bons estudos.</p>';
13     return '<html><body>' + body + '</body></html>';
14 }
15 server.listen(port, hostname, () => {
16     console.log(`Server running at http://${hostname}:${port}/`)
17 })

```

Fonte: elaborado pelo autor.

Figura 2 | Inicialização do servidor

```
C:\Program Files\nodejs>node index2.js
Server running at http://127.0.0.1:3000/
```

Fonte: elaborada pelo autor.

Figura 3 | Resultado da execução



## Oi, tudo bem?

Bons estudos.

Fonte: elaborada pelo autor.

Com isso encerramos este bloco, no qual você teve a possibilidade de conhecer a plataforma de execução para JavaScript conhecida como Node.js.

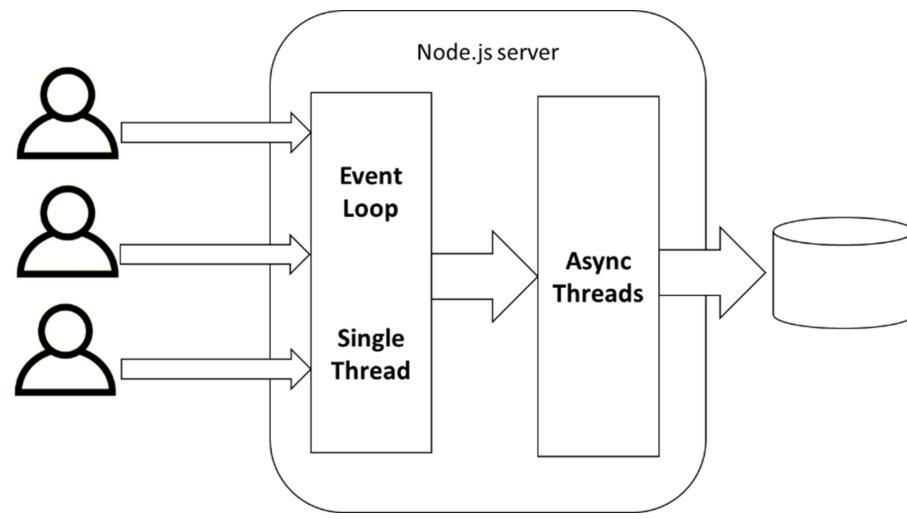
Bons estudos!

### VIDEOAULA: NODE.JS

Nesta aula você poderá conhecer mais alguns pontos sobre a arquitetura do Node.js, incluindo o modelo de uma *thread* principal de execução que atende todas as requisições externas, a criação de threads assíncronas que rodam em *background* quando necessário e a comunicação através de um loop de eventos. Vamos aprender mais a respeito?

Bons estudos!

Figura 4 | Node.js server



Fonte: elaborada pelo autor.

Para visualizar o objeto, acesse seu material digital.

### ANGULAR

Neste bloco você aprenderá sobre o Angular, um framework baseado em componentes voltado para a construção de aplicações web. Como frameworks oferecem, normalmente, um conjunto de bibliotecas para diversas necessidades de desenvolvimento, tais como roteamento, construção e gerenciamento de

formulários, comunicação cliente-servidor além de outros recursos, dispõem também de um conjunto de ferramentas para o desenvolvedor criar, compilar, testar e atualizar o código.

Angular é uma plataforma de desenvolvimento baseada em *typescript*, que, por sua vez, é uma linguagem de programação construída a partir de JavaScript. Ele não precisa de um processo de instalação para uso já que se pode simplesmente agregar Angular na escrita dos códigos das aplicações.

Esse framework pode ser usado em conjunto com Node.js, isso porque este atua no servidor e permite entregar os arquivos JavaScript necessários para o uso de Angular pelo cliente.

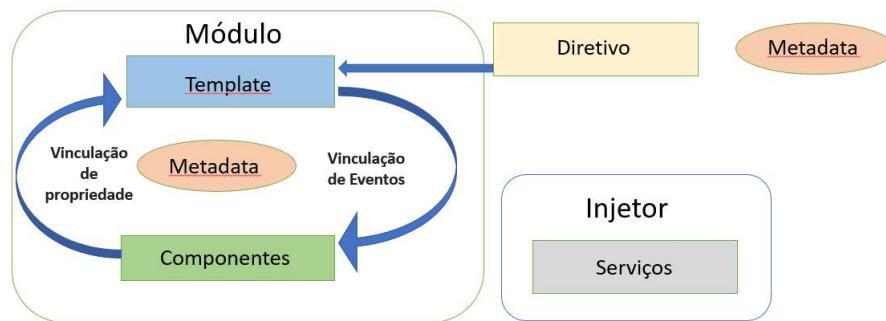
Angular é utilizado para desenvolver aplicações no lado do cliente em HTML e *typescript*. Dessa forma, pode-se construir aplicações responsivas para navegadores web ou dispositivos móveis. O forte do framework está no uso de aplicações que requeiram entregas de conteúdos dinâmicos e em tempo real.

A Figura 5 ilustra alguns dos principais recursos utilizados pelo Angular. O primeiro é o *Component*, que especifica o conjunto de elementos que podem ser construídos em uma página tal como um botão. O segundo é o *Template*, que especifica como a página é, em consequência, eventuais componentes serão renderizados. Os metadados explicitam a ligação entre esses elementos.

Angular também se baseia no uso do conceito de *dependency injection*, ou injeção de dependência. Isso significa que a flexibilidade e modularidade das aplicações aumenta, pois as classes instanciadas podem utilizar dependências de fontes externas em vez de criá-las.

Esses conceitos são também explicados em Getting... (2021).

Figura 5 | Recursos do framework Angular



Fonte: Getting... (2021, [s. p.]).

Nos Quadros 3 e 4, também disponíveis em Getting... (2021), você poderá perceber como esses conceitos são implementados. No Quadro 3, tem-se a definição de um componente associado ao arquivo template html, cujo código está no Quadro 4. Note, na linha 05 do Quadro 3, a referência ao template HTML do Quadro 4.

Quadro 3 | hello-world-bindings.component.ts

```

1 import { Component } from '@angular/core';
2
3 @Component ({
4   selector: 'hello-world-bindings',
5   templateUrl: './hello-world-bindings.component.html'
6 })
7 export class HelloWorldBindingsComponent {
8   fontColor = 'blue';
9   sayHelloId = 1;
10  canClick = false;
11  message = 'Hello, World';
12
13  sayMessage() {
14    alert(this.message);
15  }
16}

```

Fonte: Getting... (2021, [s. p.]).

Já no Quadro 4, você pode observar a declaração <button> especificando que, no evento *click*, a função *sayMessage()* deverá ser acionada. Essa funcionalidade está especificada na linha 13 do Quadro 3, que apenas exibe uma mensagem de alerta.

Ainda no Quadro 4, observe como um parágrafo pode ser construído tendo como base as propriedades do componente. *sayHelloId* (na linha 07) se refere à propriedade definida na linha 09 do Quadro 3, e a propriedade *fontColor*, na linha 11 do Quadro 4, refere-se à propriedade definida na linha 08 do Quadro 3.

Quadro 4 | hello-world-bindings.component.html

```

1  <button
2    [disabled] = "canClick"
3    (click) = "sayMessage()"
4    Trigger alert message
5  </button>
6  <p
7    [id] = "sayHelloId"
8    [style.color] = "fontColor"
9    You can set my color in the component!
10 </p>
11 <p>My color is {{ fontColor }}</p>

```

Fonte: Getting... (2021, [s. p.]).

Com esses detalhes encerramos este bloco cujo intuito foi o de oferecer uma visão mais ampla sobre o framework Angular. Considere que Angular tem muita utilização no mercado e é extremamente poderoso. Visite o site [angular.io](https://angular.io) e navegue para aprender mais detalhes sobre os recursos oferecidos.

Bons estudos!

#### **VIDEOAULA: ANGULAR**

Neste vídeo você terá a oportunidade de complementar o seu aprendizado conhecendo mais alguns detalhes sobre Angular e seus recursos. Vamos conversar mais um pouco sobre a arquitetura da plataforma e vamos explorar a ligação entre *Components* e *Templates*.

Dessa forma você terá uma percepção inicial sobre o framework e como pode ocorrer a sua utilização em projetos.

Bons estudos!

Para visualizar o objeto, acesse seu material digital.

#### **ESTUDO DE CASO**

Para contextualizar sua aprendizagem, imagine que você trabalha para uma *startup* como um desenvolvedor que faz parte de uma equipe que decidiu trabalhar com Node.js e Angular para o desenvolvimento de websites. A equipe precisa ter um treinamento sobre estes dois frameworks para poder evoluir no projeto.

Contudo, o arquiteto da equipe não poderá efetuar esse treinamento de oito horas no dia e na data planejada devido a um contratempo. Para não atrasar os trabalhos, o gerente de projetos pede para você estudar os frameworks e preparar um material de treinamento.

Como você faria para estudar e preparar essa capacitação para sua equipe? Quais temas você abordaria na ocasião?

#### **RESOLUÇÃO DO ESTUDO DE CASO**

Quando o gerente de projetos lhe fez a proposta para estudar os frameworks e, assim, realizar o treinamento com a equipe no lugar do arquiteto, ele viu em você algumas características que são importantes para enfrentar um desafio como esse, que apresenta a necessidade de um autoestudo, de conciliar as necessidades da equipe com os tópicos a serem aprendidos, além de preparar algo que seja interessante para todos os presentes.

Nesse vídeo vamos discutir alguns desses pontos para vislumbrar algum possível caminho para estruturar o treinamento, buscar informações e construir o material.

Uma possível seleção de tópicos para um treinamento sobre Node.js pode ser: introdução; primeiros passos; HTTP module; file system; NPM; eventos. Já uma possível seleção de conteúdos para um treinamento sobre Angular pode ser: introdução; recursos oferecidos; arquitetura; vantagens e limitações; exemplos de código.

Para preparar uma capacitação como essa para a equipe, você poderá buscar conteúdos de treinamentos disponibilizados na internet e adaptar algum conteúdo para a sua equipe. O importante é que seja um conteúdo preparado e adaptado para as necessidades do seu grupo de trabalho e não uma simples cópia.

Como você pensou em realizar esse treinamento? Imaginou quais aspectos nos quais você teria facilidade e quais seriam limitantes? Como resolver? Todo esse processo faz parte de um autoconhecimento e de uma aprendizagem que serão essenciais em sua vida profissional. Estar preparado para aceitar desafios como esse ajudam no desenvolvimento pessoal e profissional.

Bons estudos!

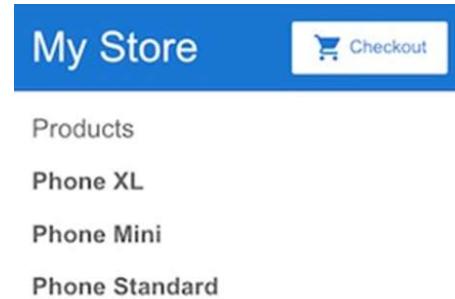
#### Resolução do Estudo de Caso

Para visualizar o objeto, acesse seu material digital.

### 6º Saiba mais

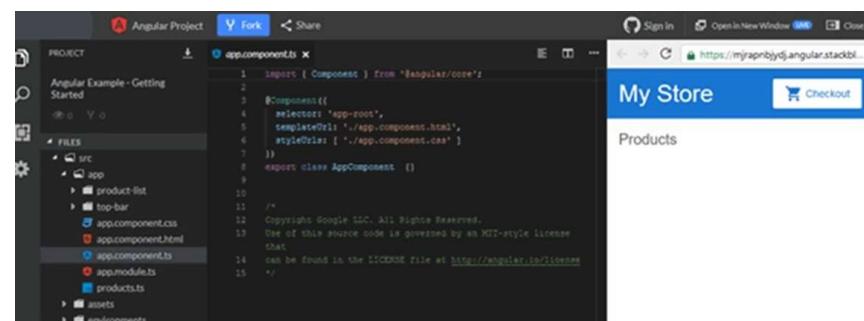
No site <https://angular.io/start> há um tutorial bem interessante para conhecer mais sobre como trabalhar com Angular. O site apresenta uma ferramenta bastante interessante chamada StackBlitz com um projeto de exemplo pronto para que você possa seguir um passo a passo e ver como a aplicação se comporta. Tudo através do seu navegador, sem precisar instalar qualquer software no seu computador.

Figura 6 | Aplicação exemplo em Angular



Fonte: Getting... (2021, [s. p.]).

Figura 7 | Ambiente StackBlitz



Fonte: Getting... (2021, [s. p.]).

**Aula 3****FRAMEWORKS PHP**

Nesta aula você terá a oportunidade de aprender a arquitetura geral e o propósito de alguns frameworks bastante utilizados com PHP no mercado, tais como: Laravel, Symfony, CodeIgniter e Zend.

44 minutos

**INTRODUÇÃO**

Nesta aula você terá a oportunidade de aprender a arquitetura geral e o propósito de alguns frameworks bastante utilizados com PHP no mercado, tais como: Laravel, Symfony, CodeIgniter e Zend.

Além disso, você estudará os frameworks Laravel e Symfony com mais detalhes, com isso terá um conhecimento abrangente dos frameworks usados com PHP e como trabalhar com eles.

Em sua vida profissional, possivelmente você terá contato com projetos baseados em PHP e com esses frameworks. Desse modo, espera-se que você aproveite esta oportunidade para evoluir no conhecimento sobre os principais frameworks utilizados no mercado.

Bons estudos!

**FRAMEWORKS PARA PHP**

Muitos frameworks existem para se trabalhar com PHP; alguns adotam um caráter mais generalista, procurando atender tanto as camadas de apresentação quanto as de lógica de negócios, e outros procuram atender aspectos específicos do desenvolvimento. Veja alguns deles no Quadro 1.

Quadro 1 | Frameworks PHP

Nome do framework	Site	Licença
CodeIgniter	<a href="http://codeigniter.com">codeigniter.com</a>	MIT
Laminas ( <i>former Zend</i> )	<a href="http://getlaminas.org">getlaminas.org</a>	New BSD
Laravel	<a href="http://laravel.com">laravel.com</a>	MIT
Symfony	<a href="http://symfony.com">symfony.com</a>	MIT

Fonte: elaborado pelo autor.

Em 2009, foi criado o *PHP Framework Interop Group* ou PHP-FIG. Esse grupo promove recomendações, na forma de PSR ou *PHP Standard Recommendations*, com o intuito de promover colaboração e interoperabilidade entre os frameworks. Dessa forma, a intenção é que cada framework possa indicar quais recomendações segue e dar maior segurança aos desenvolvedores em relação ao que se está usando em determinado projeto. Vamos conhecer os frameworks citados no Quadro 1?

**CodeIgniter**

Oferece um kit de ferramentas para se construir websites com PHP. Tem como objetivo acelerar o desenvolvimento a partir de um conjunto de bibliotecas as quais automatizam uma série de atividades que um desenvolvedor consumiria muito tempo escrevendo.

No Quadro 2, você poderá ver o exemplo de um *Controller* criado com o CodeIgniter, conforme explicado em CodeIgniter (2022). O *controller* é uma classe centralizadora das requisições para a aplicação web a partir da qual você poderá criar outros *templates* de páginas web e agregar outras funcionalidades, como trabalhar com bancos de dados.

Quadro 2 | Exemplo de Controller criado com CodeIgniter

```

1 <?php
2 namespace App\Controllers;
3 class Pages extends BaseController
4 {
5     public function index()
6     {
7         return view('welcome_message');
8     }
9     public function view($page = 'home')
10    {
11        // ...
12    }
13 }
14 }
```

Fonte: CodeIgniter (2022).

## Laminas

Esse projeto é uma evolução *open-source* do framework Zend e tem por finalidade oferecer um conjunto de bibliotecas de componentes para suportar aplicações empresariais.

Um dos projetos que fazem parte do framework é o Laminas Components, o qual contempla uma série de componentes como *dependency injection*, filtros, validadores, internacionalização, mensagens, navegação, paginação e outros. Já o projeto Laminas MVC contempla uma camada de tratamento de eventos e permite a construção de aplicações MVC, controladores (*controllers*) e *plugins*.

Outro projeto que faz parte do framework é o Mezzio, um ambiente de execução PHP (ou *middleware*) com vários componentes e extensões. Apresenta similaridade de objetos com Laminas MVC, contudo oferece uma estrutura minimalista, a qual resulta em uma menor curva de aprendizado. Por fim, faz parte também do framework a Laminas API Tools, que é um construtor de APIs, ou seja, permite que um desenvolvedor construa, teste e mantenha suas próprias APIs, controlando e gerenciando acesso por vários consumidores.

Figura 1 | Projetos Laminas



Fonte: Laminas (c2020, [s. p.]).

## Laravel

Framerwork com foco no desenvolvimento de aplicações para a web, possui uma extensa biblioteca de funcionalidades e um site com bastante conteúdo, incluindo guias e tutoriais. Também engloba ferramentas para injeção de dependências, testes unitários e tratamento de eventos em tempo real. Possui uma comunidade bem desenvolvida no mercado.

## Symfony

Engloba o Simfony Framework, que é utilizado para criar websites e aplicações a partir do Symfony Components, os quais são componentes desacoplados e reutilizáveis. Além disso, ainda de acordo com Symfony (2022), engloba uma comunidade com mais de 600.000 desenvolvedores e uma filosofia de promoção de boas práticas, padronização e interoperabilidade. As bibliotecas contemplam integração com bancos de dados, mensageria, troca de mensagens, internacionalização, segurança, apresentação, *workflows* e muitos outros recursos.

## Conclusão

Assim concluímos esta aula: com uma visão geral dos frameworks para PHP disponíveis no mercado. Vale ressaltar que ainda existem vários outros para os mais variados propósitos e que os citados nesta aula têm destaque e histórico no mercado profissional.

Para complementar seu conhecimento, visite alguns dos sites mencionados dos projetos e navegue na documentação para obter mais informações.

Bons estudos!

## VIDEOAULA: FRAMEWORKS PARA PHP

Neste vídeo discutiremos sobre alguns frameworks para PHP utilizados no mercado. Muitos deles focam no desenvolvimento web, mas outros também disponibilizam recursos para diferentes focos de uso, como cenários de IoT e Big Data por exemplo. Vale a pena saber que existem vários frameworks e entender como você poderá escolhê-los para usar em algum projeto.

Videoaula: Frameworks para PHP

Para visualizar o objeto, acesse seu material digital.

### LARAVEL

Neste bloco vamos estudar o Laravel, um dos frameworks para PHP com maior utilização no mercado. No caso dele, existem vários recursos que facilitam a criação de aplicações e uma clara separação de responsabilidades, as quais buscam estruturar uma arquitetura MVC. As informações sobre o projeto, incluindo documentação e procedimentos para instalação e configuração, estão em <https://laravel.com/>.

O Laravel é um framework *open-source* que, em alguns casos, também atua como um *middleware*. Surgiu em 2011 e oferece vários recursos para administração e desenvolvimento de aplicações tais como: *eloquent ORM*; *query builder*; *reverse routing*; *restful controllers*; *unit testing*, entre outros. Também tem pacotes que atendem necessidades como cobrança ou assinaturas, infraestrutura, autenticação e autorização, busca textual e autenticação com redes sociais.

O framework requer o ambiente do PHP, do *Composer* (que é um gerenciador de dependências para as aplicações escritas com PHP) e do Docker. Algumas instruções de instalação e configuração variam entre ambientes como Linux, Windows e Mac.

Um procedimento para criar a estrutura de diretórios com uma aplicação PHP construída com Laravel é executar as instruções do Quadro 3 na linha de comando do seu sistema operacional Linux. Na linha 01, um *script* é descarregado do link indicado e executado na máquina para construir uma aplicação (com o nome *example-app*, nesse caso) com arquivos e toda a estrutura de diretórios montada. Na linha 02, entra-se na pasta do projeto e, na linha 03, aciona-se e inicializa-se o ambiente para execução do projeto. O programa *sail* inicializa o ambiente Docker com o Laravel em execução.

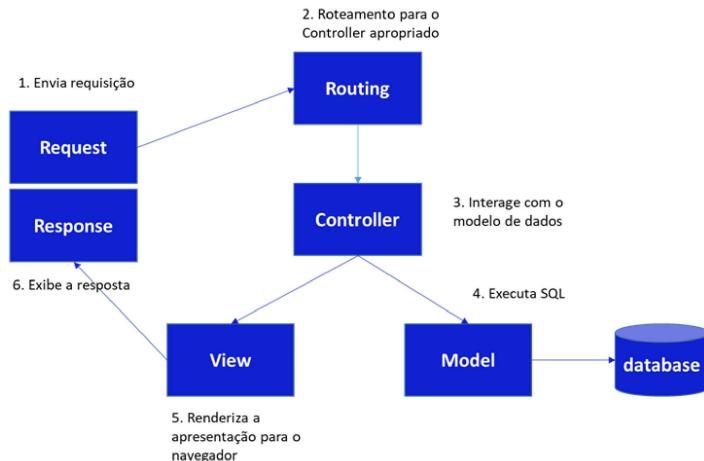
Quadro 3 | Criação e inicialização de uma aplicação com Laravel

```
1 curl -s https://laravel.build/example-app | bash
2 cd example-app
3 ./vendor/bin/sail up
```

Fonte: Laravel (c2022, [s. p.]).

Você pode observar, na Figura 2, a visão geral da arquitetura, que contempla alguns dos componentes do Laravel. Como um framework MVC, as responsabilidades estão divididas entre componentes cujos códigos se dividem entre as camadas de *Model*, *View* e *Controller*. Quando uma requisição é enviada, ela é capturada pelo mecanismo de roteamento estruturado pelo Laravel e direcionada ao *Controller* apropriado para tratamento. O código no *Controller* buscará encaminhar e controlar as instruções para a camada *Model* e devolver o conteúdo necessário para a camada *View*. Com todo esse processo concluído, a resposta é encaminhada para o solicitante.

Figura 2 | Visão geral da arquitetura Laravel



Fonte: adaptada de Budiman et al. (2017).

Você deve estar curioso sobre como ficam os códigos com esse framework. No próprio site do Laravel, existem diversos tutoriais e códigos para aprendizado com exemplos mais sofisticados. Mas, para iniciar este estudo, observe os Quadros 4 e 5.

Quadro 4 | Exemplo de código View com Laravel

```

1 <!-- View stored in resources/views/greeting.blade.php -->
2 <html>
3   <body>
4     <h1>Hello, {{ $name }}</h1>
5   </body>
6 </html>
  
```

Fonte: Views ([s. d., s. p.]).

No Quadro 4, o principal aspecto é a obtenção do valor da variável \$name para exibição. E, no Quadro 5, destaca-se como é possível atribuir um valor para a variável (no caso, um valor estático).

Quadro 5 | Exemplo de código View Helper com Laravel

```

1 Route::get('/', function () {
2   return view('greeting', ['name' => 'James']);
3 });
  
```

Fonte: Views ([s. d., s. p.]).

No Quadro 6, você pode observar a criação de um Controller chamado UserController. Repare na declaração da linha 08 (onde UserController estende uma classe do framework chamada Controller). Em seguida, repare, na linha 16, a criação de uma ação (show) para buscar o perfil de um usuário.

Quadro 6 | Exemplo de código Controller com Laravel

```

1 <?php
2
3 namespace App\Http\Controllers;
4
5 use App\Http\Controllers\Controller;
6 use App\Models\User;
7
8 class UserController extends Controller
9 {
10
11     /**
12      * Show the profile for a given user.
13      *
14      * @param int $id
15      * @return \Illuminate\View\View
16      */
17     public function show($id)
18     {
19         return view('user.profile', [
20             'user' => User::findOrFail($id)
21         ]);
22     }

```

Fonte: Controllers ([s. d., s. p.]).

No Quadro 7, você poderá observar o mapeamento entre uma URL e a funcionalidade declarada no Quadro 6. Na linha 03, você pode observar que o caminho /user/id será direcionado para o método *show*.

Quadro 7 | Exemplo de código de roteamento com Laravel

```

1 use App\Http\Controllers\UserController;
2
3 Route::get('/user/{id}', [UserController::class, 'show']);

```

Fonte: Controllers ([s. d., s. p.]).

Já no Quadro 8, ocorre a declaração de uma entidade. Veja a linha 07 com a classe *Flight* (que representa uma entidade) como extensão da classe *Model* do framework. Na linha 14, há também a declaração de uma chave primária. Existem regras no framework para mapear automaticamente a classe e as propriedades com o banco de dados, mas você pode declarar manualmente se necessário.

Quadro 8 | Exemplo de código *Model* com Laravel

```

1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Model;
6
7 class Flight extends Model
8 {
9
10     /**
11      * The primary key associated with the table.
12      *
13      * @var string
14      */
15     protected $primaryKey = 'flight_id';

```

Fonte: Eloquent... ([s. d., s. p.]).

Por fim, no Quadro 9, você pode observar o exemplo de uma consulta de todos os *Flights* da tabela. Vale ressaltar que é possível usar funções definidas pelo framework para automaticamente acessar os dados, mas também é possível especificar ou declarar SQL se for preciso.

Quadro 9 | Exemplo de código *Model* de consulta com Laravel

```

1  use App\Models\Flight;
2
3  foreach (Flight::all() as $flight) {
4      echo $flight->name;
5  }

```

Fonte: Eloquent... ([s. d., s. p.]).

Assim encerramos este bloco sobre Laravel. O objetivo é que você possa ter tido uma visão geral sobre o framework para compreender como ele se encaixa ao mundo PHP. Novamente, fica a recomendação para que você realize a navegação pelo conteúdo do site do projeto a fim de ampliar seus conhecimentos.

#### **VIDEOAULA: LARAVEL**

Nesta aula você poderá conhecer mais alguns pontos sobre a arquitetura do Laravel, sua estruturação e organização baseada em MVC, bem como alguns detalhes sobre o conteúdo apresentado neste bloco. Você também reverá os exemplos de código discutidos em aula.

Vamos aprender mais alguns conceitos sobre o Laravel?

Videoaula: Laravel

Para visualizar o objeto, acesse seu material digital.

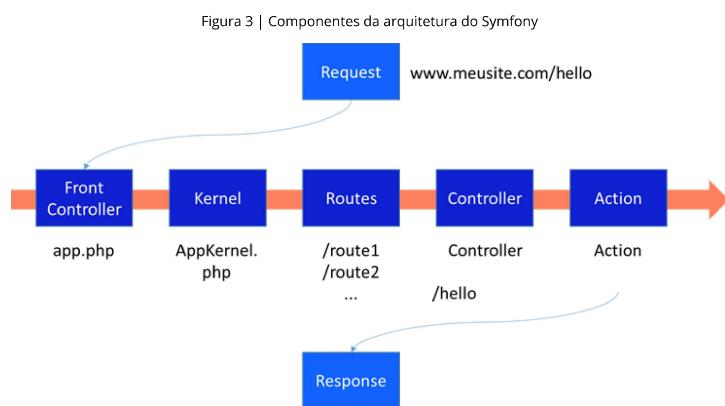
#### **SYMFONY**

A partir de agora você conhecerá o framework Symfony. Para instalá-lo, a recomendação via documentação, presente em <https://symfony.com>, é usar o gerenciador de dependências *Composer*, como o mostrado a seguir:

```
composer require symfony/all-my-sms-notifier
```

Symfony possui um conjunto de componentes para serem utilizados. Eles são bibliotecas desacopladas e já foram testados e usados em diversos projetos. De acordo com Projects... ([s. d.]), existem vários projetos que usam o Symfony, alguns deles são: Drupal, Joomla, Magento, Laravel, Google APIs, Facebook Ads APIs e muitos outros.

Para entender como Symfony atua, observe a Figura 3. Uma requisição é realizada (por exemplo, para a URL <http://www.meusite.com/hello>) e é encaminhada para o *Front Controller*, que, por sua vez, instanciará o Kernel. Dessa avaliação a requisição será então roteada para o *Controller* apropriado e, de lá, para o *Action* requerido. Dessa forma, a resposta é construída e encaminhada para o navegador.



Fonte: adaptada de Introduction (2022, [s. p.])

**Front Controller** é um design *pattern* que recebe todas as requisições da aplicação. No Symfony esse é o papel do arquivo index.php e é o primeiro script executado quando a requisição é processada. O papel principal dele é criar uma instância do Kernel para manipular a requisição e retornar a resposta.

- **Kernel** é o core do Symfony e é responsável por configurar os recursos usados. Cria o service container e o envia para processamento.

- **Routes** faz a avaliação das rotas de acordo com as definições feitas.
- **Controller** recebe o objeto *Request* e constrói ou devolve um objeto *Response*.
- **Action** é um método ou ação chamado pelo *Controller* para o processamento.

No Quadro 10, você poderá observar um exemplo inicial de um código escrito com Symfony. Neste código se cria um *Controller* que pode ser associado com uma URL, conforme ilustrado no Quadro 11. O código em si, em conjunto, exibe uma página com um número aleatório.

Quadro 10 | Exemplo de código Symfony

```

1 <?php
2 // src/Controller/LuckyController.php
3 namespace App\Controller;
4
5 use Symfony\Component\HttpFoundation\Response;
6
7 class LuckyController
8 {
9     public function number(): Response
10    {
11        $number = random_int(0, 100);
12
13        return new Response(
14            '<html><body>Lucky number: ' . $number . '</body></html>';
15        );
16    }
17 }
```

Fonte: Create... (2022, [s. p.]).

No Quadro 11, tem-se a associação de uma URL (/Lucky/number) com o código do *Controller* LuckyController.

Quadro 11 | Exemplo de roteamento Symfony

```

1 # config/routes.yaml
2
3 # the "app_lucky_number" route name is not important yet
4 app_lucky_number:
5     path: /lucky/number
6     controller: App\Controller\LuckyController::number
```

Fonte: Create... (2022, [s. p.]).

Symfony também usa Twig, uma linguagem poderosa para escrever HTML. Um uso análogo ao exibido no Quadro 10 poderia ser escrito da seguinte forma:

```
{# templates/lucky/number.html.twig #-}

Your lucky number is {{ number }}
```

Para comunicação com banco de dados, Symfony usa Doctrine, um conjunto de bibliotecas que trabalha com bancos de dados relacionais (como MySQL ou PostgreSQL) e bancos de dados NoSQL como MongoDB.

No Quadro 12, você pode ver um exemplo de uma entidade para uso com o Symfony através de Doctrine. O exemplo cria uma classe chamada *Product* com uma chave id, o nome e o preço. A biblioteca Doctrine fará o tratamento necessário para que um objeto esteja relacionado com uma linha do banco de dados na tabela.

Quadro 12 | Exemplo de uma classe de entidade no Symfony

```

1 // src/Entity/Product.php
2 namespace App\Entity;
3
4 use App\Repository\ProductRepository;
5 use Doctrine\ORM\Mapping as ORM;
6
7 /**
8 * @ORM\Entity(repositoryClass=ProductRepository::class)
9 */
10 class Product
11 {
12     /**
13     * @ORM\Id()
14     * @ORM\GeneratedValue()
15     * @ORM\Column(type="integer")
16     */
17     private $id;
18
19     /**
20     * @ORM\Column(type="string", length=255)
21     */
22     private $name;
23
24     /**
25     * @ORM\Column(type="integer")
26     */
27     private $price;
28
29     public function getId(): ?int
30     {
31         return $this->id;
32     }
33
34     // ... getter and setter methods
35 }
```

Fonte: Databases... (2022, [s. p.]).

Com a biblioteca, outras funcionalidades para criação de entidades, tabelas, operações sobre linhas no banco de dados e consultas SQLs serão manipuladas através dos recursos Doctrine.

Assim concluímos esta aula: oferecendo uma visão geral sobre o Symfony e sobre os recursos oferecidos por ele. Aproveite o momento para visitar o site do projeto, pois existem vários recursos que podem ser explorados com esse framework. Vale a pena também conhecer a documentação, instalar e executar alguns exemplos.

Bons estudos!

#### **VIDEOAULA: SYMFONY**

Symfony é um framework bastante poderoso e com uma série de recursos para a criação de aplicações web. Nesta videoaula você poderá conhecer mais alguns pontos e detalhes a respeito dele e se preparar para uma futura discussão ou imersão sobre o framework. Além disso, poderá perceber eventuais diferenças e semelhanças entre distintos frameworks para ter uma visão mais precisa sobre qual será útil em determinado projeto.

Videoaula: Symfony

Para visualizar o objeto, acesse seu material digital.

#### **ESTUDO DE CASO**

Você trabalha em uma equipe que desenvolve com PHP sem o uso de *middleware* ou de frameworks conhecidos. Ou seja, o PHP puro.

Você tem a certeza de que o uso de frameworks pode agilizar os trabalhos dessa equipe, mas o time em si discorda por já estarem acostumados com aquela forma de trabalhar e terem receio de incorporar componentes externos ao desenvolvimento.

Como você faria para argumentar, demonstrar ou provar que o uso de frameworks para PHP como Laravel ou Symfony serão benéficos para a equipe?

### RESOLUÇÃO DO ESTUDO DE CASO

A situação descrita no estudo de caso é bastante comum de se encontrar em equipes de desenvolvimento de empresas pequenas, médias ou grandes. Em geral, a equipe acaba tendo uma forma de trabalhar que pode tornar difícil a adoção de novas tecnologias.

Na situação em questão, a intenção é que você pense como mostrar para os integrantes do grupo que o uso de Laravel ou Symfony pode ser interessante. Mas, para isso, talvez você precise pensar em algumas alternativas: fazer um treinamento; fazer uma defesa técnica perante o time; demonstrar as diferenças e ganhos trazidos pelo framework.

Portanto, vale a pena você visitar este link, que o redirecionará para uma página na qual há uma comparação entre o uso do Symfony em relação a PHP:

[https://symfony.com/doc/current/introduction/from\\_flat\\_php\\_to\\_symfony.html](https://symfony.com/doc/current/introduction/from_flat_php_to_symfony.html)

Resolução do Estudo de Caso

Para visualizar o objeto, acesse seu material digital.

### 6º Saiba mais

#### Docker

Você deve ter reparado que na aula sobre Laravel foi mencionado o uso do software Docker. Você já o conhece?

Docker é uma plataforma aberta para desenvolver, distribuir e executar aplicações. Ele separa a aplicação da infraestrutura e permite empacotar e rodar a aplicação em um ambiente isolado chamado *container*.

Veja mais em: <https://www.docker.com/>. Acesso em 11 fev. 2022.

#### Kubernetes

Se achar interessante conhecer Docker, valerá a pena também conhecer uma plataforma de orquestração de contêineres Docker.

Veja mais em: <https://kubernetes.io/pt-br/>. Acesso em 11 fev. 2022.

#### Laravel e Symfony

No bloco sobre Symfony, foi citado que Laravel também usa alguns componentes criados para o framework. Isso reforça a visão de que os frameworks possibilitam a criação de outras aplicações, mas também possibilitam a criação de novos frameworks.

Veja mais em: <https://symfony.com/projects/laravel>. Acesso em 11 fev. 2022.

**Aula 4****FRAMEWORKS PYTHON**

*Nesta unidade você terá a oportunidade de aprender sobre a arquitetura geral e o propósito de alguns frameworks bastante utilizados com Python no mercado.*

*45 minutos*

**INTRODUÇÃO**

Nesta unidade você terá a oportunidade de aprender sobre a arquitetura geral e o propósito de alguns frameworks bastante utilizados com Python no mercado. Alguns deles são: Django, Flask, Web2py, CherryPy, Bottle, entre outros. Dois deles, Django e Flask, serão estudados aqui com mais detalhes. Com isso, você terá um conhecimento abrangente dos frameworks usados com Python e de como trabalhar com eles.

Em sua vida profissional, possivelmente, você terá contato com projetos baseados em Python e com alguns desses frameworks. Espera-se, portanto, que você aproveite esta oportunidade para evoluir no conhecimento sobre os principais frameworks utilizados no mercado.

Bons estudos!

**FRAMEWORKS PARA PYTHON**

Nesta aula você terá uma visão ampla sobre os frameworks utilizados na linguagem Python. Assim, você aprenderá sobre como esses frameworks são classificados e terá uma visão geral sobre alguns deles. Vamos começar?

Em um primeiro momento, vale dizer que a linguagem Python é bastante poderosa e simplifica demais o trabalho do desenvolvedor. Contudo, quando se começa a pensar em desenvolver algum tipo de aplicação, é importante ter facilitadores para o processo, ou seja, ferramentas cujo intuito seja o de facilitar a criação de formulários, as ações de autenticação ou autorização em páginas, a conectividade com bancos de dados, entre outros.

Existe um conjunto bem extenso de bibliotecas e de frameworks que podem ser utilizados e o mais interessante é que muitas delas se encontram catalogadas no *Python Package Installer* (PyPI – acessível via <https://pypi.org/>). E este será, então, o nosso ponto de partida para eventuais referências e acesso aos projetos. Além disso, cada projeto listado pode ser instalado por meio do pip (ou *Package Installer for Python*, instalador de pacotes para Python).

Os frameworks em Python são classificados em três tipos: *full-stack*, microframeworks e assíncronos. Vamos aprender sobre cada um deles?

**Frameworks *full-stack***

Esse tipo de framework é empacotado com um extenso conjunto de funcionalidades. O objetivo dele é já contemplar um grande conjunto de necessidades que os desenvolvedores apresentem para construir aplicações. Dessa forma, o que se busca é oferecer um conjunto amplo de funcionalidades que atenderão o desenvolvimento desde criação de interfaces até procedimentos de segurança, lógica de negócios e conectividade com bancos de dados. No Quadro 1, você pode visualizar alguns desses frameworks.

Quadro 1 | *Full-stack* frameworks para Python

Nome	Site	Licença
CubicWeb	<a href="https://pypi.org/project/cubicweb/">https://pypi.org/project/cubicweb/</a>	LGPL
Django	<a href="https://pypi.org/project/Django/">https://pypi.org/project/Django/</a>	BSD

Nome	Site	Licença
Giotto	<a href="https://pypi.org/project/giotto/">https://pypi.org/project/giotto/</a>	BSD-2-Clause
Pylons	<a href="https://pypi.org/project/Pylons/">https://pypi.org/project/Pylons/</a>	BSD
Pyramid	<a href="https://pypi.org/project/pyramid/">https://pypi.org/project/pyramid/</a>	Repoze
TurboGears	<a href="https://pypi.org/project/TurboGears/">https://pypi.org/project/TurboGears/</a>	MIT
Web2Py	<a href="https://pypi.org/project/web2py/">https://pypi.org/project/web2py/</a>	LGPL

Fonte: Singh (2022, [s. p.]).

Da lista no Quadro 1, vamos conversar sobre dois deles: Django e Web2Py, ambos bastante conhecidos e utilizados.

**Django** segue o princípio DRY (*don't repeat yourself*, ou, não se repita). Possui um grande conjunto de funcionalidades oferecidas como parte do framework e não como um conjunto de bibliotecas agrupadas. Por exemplo, Django faz uso de um mecanismo próprio para mapeamento objeto-relacional.

**Web2Py** tem uma abordagem ampla também e possui um ambiente integrado de desenvolvimento próprio que conta com editor de código, debugador e desenvolvimento rápido. Apresenta um mecanismo para MVC, recursos para conectividade com banco de dados e internacionalização e localização.

### Microframeworks

Os microframeworks podem ser soluções que atuam em um domínio específico de problema ou que podem apresentar um conjunto bastante amplo de recursos, porém a instalação inicial é geralmente pequena, com alguns componentes básicos. Caso o desenvolvedor precise de mais recursos, eles podem ser agregados conforme a necessidade. Os microframeworks em geral não obrigam a forma de trabalho para um desenvolvedor. Veja, no Quadro 2, alguns exemplos.

Quadro 2 | Microframeworks para Python

Nome	Site	Licença
Bottle	<a href="https://pypi.org/project/bottle/">https://pypi.org/project/bottle/</a>	MIT
CherryPy	<a href="https://pypi.org/project/CherryPy/">https://pypi.org/project/CherryPy/</a>	BSD
Dash	<a href="https://pypi.org/project/dash/">https://pypi.org/project/dash/</a>	MIT
Falcon	<a href="https://pypi.org/project/falcon/">https://pypi.org/project/falcon/</a>	ASL 2.0
Flask	<a href="https://pypi.org/project/Flask/">https://pypi.org/project/Flask/</a>	BSD-3-Clause
Hug	<a href="https://pypi.org/project/hug/">https://pypi.org/project/hug/</a>	MIT
MorePath	<a href="https://pypi.org/project/morepath/">https://pypi.org/project/morepath/</a>	BSD
Pycnic	<a href="https://pypi.org/project/pycnic/">https://pypi.org/project/pycnic/</a>	MIT

Fonte: Singh (2022, [s. p.]).

Da lista no Quadro 2, vamos conversar sobre Bottle, CherryPy e Flask, que são microframeworks bastante conhecidos e utilizados no mercado.

**Bottle** foi originalmente desenvolvido para construir APIs, mas agora já possibilita também a construção de aplicações web. Entrega uma série de recursos em um único arquivo, o qual contempla roteamento, *templates*, utilitários e um servidor HTTP.

**CherryPy** segue uma abordagem minimalista e entrega uma aplicação Python com um servidor web munido de recursos para conectividade com banco de dados e vários recursos nativos.

**Flask** também permite o uso de vários recursos, mas inicialmente oferece um conjunto mínimo para escrever aplicações web. Com o recurso de extensões, as funcionalidades utilizadas podem ser expandidas conforme as necessidades da aplicação.

### Frameworks assíncronos

Os frameworks assíncronos entregam recursos para construção de código concorrente e fazem uso da biblioteca `asyncio`. Veja em <https://docs.python.org/3/library/asyncio.html> mais informações sobre essa biblioteca e, no Quadro 3, alguns frameworks com essa característica.

Quadro 3 | Frameworks assíncronos para Python

Nome	Site	Licença
AIOHTTP	<a href="https://pypi.org/project/aiohttp/">https://pypi.org/project/aiohttp/</a>	ASL 2.0
Growler	<a href="https://pypi.org/project/growler/">https://pypi.org/project/growler/</a>	ASL 2.0
Sanic	<a href="https://pypi.org/project/sanic/">https://pypi.org/project/sanic/</a>	MIT
Tornado	<a href="https://pypi.org/project/tornado/">https://pypi.org/project/tornado/</a>	ASL 2.0

Fonte: Singh (2022, [s.p.]).

Com isso encerramos esta aula referente à visão geral dos frameworks para Python. Visite o site *PyPI* para conhecer alguns dos projetos listados. Para complementar seus estudos, os próximos blocos abordarão alguns detalhes adicionais sobre Django e Flask.

Bons estudos!

### VIDEOAULA: FRAMEWORKS PARA PYTHON

Nesta aula o objetivo é ter uma visão abrangente sobre os frameworks para Python. Por isso, neste vídeo você poderá reforçar seus conhecimentos sobre os tipos de frameworks e conhecer alguns deles. Ao final, você estará preparado para diferenciar os recursos providos pelos diferentes frameworks. Bons estudos!

Videoaula: Frameworks para Python

Para visualizar o objeto, acesse seu material digital.

### DJANGO

O Django é um framework para a linguagem Python de bastante importância. Foi criado ainda em 2003 por dois desenvolvedores que trabalhavam na construção de um site de notícias, inicialmente, com PHP e, depois, com Python. Assim, eles se propuseram a criar um framework para a construção de aplicações web.

Informações sobre o projeto podem ser encontradas em <https://pypi.org/project/Django> ou em <https://www.djangoproject.com/>. Django é um framework classificado como *full-stack*, pois permeia várias funcionalidades tanto para as camadas de apresentação quanto para as de controle e persistência. Essas funcionalidades estão agrupadas em itens como: modelo, apresentação, *templates*, desenvolvimento, administração, segurança, internacionalização e localização, performance e otimização, georreferenciamento, ferramentas web, entre outros recursos.

Vale também mencionar que os mecanismos presentes no Django não são um mero agrupamento de bibliotecas, mas funcionalidades desenvolvidas especificamente para uso com os elementos arquiteturais presente nesse framework.

A instalação do Django é um processo bem simplificado. Basta executar:

```
python -m pip install Django
```

A partir disso, você pode criar um projeto, o que é possível também através da linha de comando com django-admin:

```
django-admin startproject mywebsite
```

Esse comando criará uma estrutura de pastas incluindo arquivos iniciais necessários para o projeto. A partir daí basta escrever, adicionar o código e ativar o servidor web.

Para que você tenha uma visão de como é um código em Python escrito com Django, observe os Quadros 4, 5 e 6. Neles estão apresentados alguns códigos que terão como objetivo gerar o código HTML contido no Quadro 7. Mais detalhes e explicações sobre o código podem ser encontrados em Django (c2022), que apresenta um tutorial para a criação de formulários e no qual há códigos que serão discutidos em maior profundidade.

Agora, note a linha 03 do Quadro 4. Ali se explicita a definição de um campo de formulário chamado *your\_name*. Já o código no Quadro 5 indica a construção do formulário e, na linha 02, o *token* CSRF indica um valor secreto para o site e é usado como uma proteção de segurança.

Quadro 4 | Exemplo de código forms.py

```
1 from django import forms
2 class NameForm(forms.Form):
3     your_name = forms.CharField(label='Your name', max_length=100)
```

Fonte: Django (c2022, [s. p.]).

Quadro 5 | Exemplo de código name.html

```
1 <form action="/your-name/" method="post">
2     {% csrf_token %}
3     {{ form }}
4     <input type="submit" value="Submit">
5 </form>
```

Fonte: Django (c2022, [s. p.]).

O Quadro 6 traz o código de apresentação e renderização do formulário e a associação com o formulário NameForm do Quadro 4. O código basicamente recebe a requisição e processa a apresentação ou o envio para uma página de agradecimento.

Quadro 6 | Exemplo de código views.py

```
1 from django.http import HttpResponseRedirect
2 from django.shortcuts import render
3 from .forms import NameForm
4 def get_name(request):
5     if request.method == 'POST':
6         form = NameForm(request.POST)
7         if form.is_valid():
8             return HttpResponseRedirect('/thanks/')
9     else:
10         form = NameForm()
11     return render(request, 'name.html', {'form': form})
```

Fonte: Django (c2022, [s. p.]).

O código apresentado no Quadro 7 representa o objetivo que se buscou atingir com esses arquivos: possibilitar-lhe o entendimento de como o Django é utilizado e de como pode ajudar na criação das páginas da web.

Quadro 7 | Exemplo de código name.html

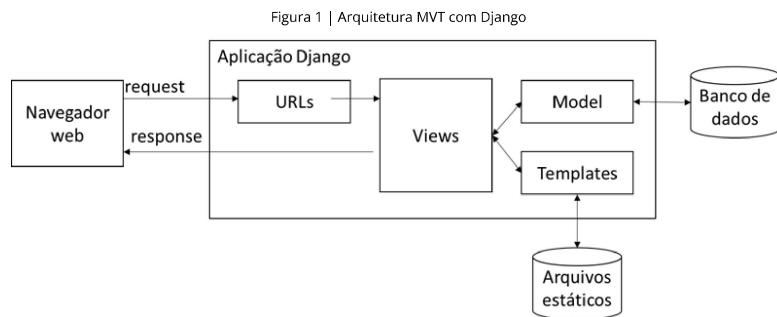
```

1 <form action="/your-name/" method="post">
2 <label for="your_name">Your name: </label>
3 <input id="your_name" type="text" name="your_name" value="{{ current_name }}">
4 <input type="submit" value="OK">
5 </form>

```

Fonte: Django (c2022, [s. p.]).

Django é baseado na arquitetura MVT (ou *Model-View-Template*), tal como ilustra a Figura 1. Nessa arquitetura, a camada *Model* é responsável por manter os dados e também interagir com o banco de dados. A camada *View* é a interface com o usuário, e a camada *Template* diz respeito aos arquivos estáticos do HTML a ser gerado. A principal diferença reside no fato de que os elementos de controle da camada *Controller* são tratados pelo próprio framework. Por essa razão, explicita-se essa diferenciação entre as arquiteturas. Na prática, seu código pode seguir um modelo MVT ou um MVC, sem restrições.



Fonte: adaptada de Bioco e Rocha (2019).

Com isso chegamos ao final desta aula. Espera-se que ela o tenha feito se interessar por conhecer mais sobre o framework Django, um framework estável, pois foi usado e testado em uma variedade de projetos. Existem ainda vários recursos que foram construídos pelo time responsável e que merecem ser pesquisados. Vá em frente e bons estudos!

### VIDEOAULA: DJANGO

Django é um framework *full-stack* de grande importância no mundo Python. Tendo sido criado em 2003, evoluiu muito ao longo dos anos e fomentou uma grande comunidade e uma variedade de recursos que favorecem a criação de aplicações web. Neste vídeo você vai ter uma visão geral sobre esse framework.

Videoaula: Django

Para visualizar o objeto, acesse seu material digital.

### FLASK

Flask é um framework para Python com um longo histórico: foi criado em 2004 e é considerado um microframework. O funcionamento básico é mapear URLs para o código de execução ou, em outras palavras, manipular requisições e retornar o HTML correspondente.

Dessa forma, a arquitetura é simples, porém extensível. Recursos como enviar mensagens por e-mail ou mesmo se conectar a um banco de dados não fazem parte do framework, mas podem ser utilizados a partir de extensões, chamadas de Foo-Flask ou Flask-Foo.

Em sendo um microsserviço, a instalação e a construção de uma aplicação inicial são realizadas de forma bem simples e ágil. Para fazer a instalação basta usar:

```
$ pip install Flask
```

Para construir uma aplicação, o Quadro 8 ilustra um código que exibe uma mensagem Hello. O funcionamento envolve exibir *Hello World*, porém, se um parâmetro *name* for preenchido, a saída será *Hello* seguida pelo valor de *name*.

Quadro 8 | Exemplo de código app.py com Flask

```

1  from flask import Flask, escape, request
2  app = Flask(__name__)
3  @app.route('/')
4  def hello():
5      name = request.args.get("name", "World")
6      return f'Hello, {escape(name)}!'

```

Fonte: Flask (c2022).

O código contido no Quadro 8, salvo como app.py, pode simplesmente ser executado acionando:

```
$ flask run
```

Com isso, o servidor web do Flask estará atendendo requisições na porta 5000. Logo, pode-se acessar a página informando a URL <http://127.0.0.1:5000/> que o resultado exibido será *Hello World*. Porém, ao se passar um valor para o parâmetro *name*, esse conteúdo será usado para a mensagem. Na Figura 2, tem-se a saída realizada ao se atribuir o texto **estudante** para a variável *name* via URL.

Figura 2 | Execução de app.py com Flask

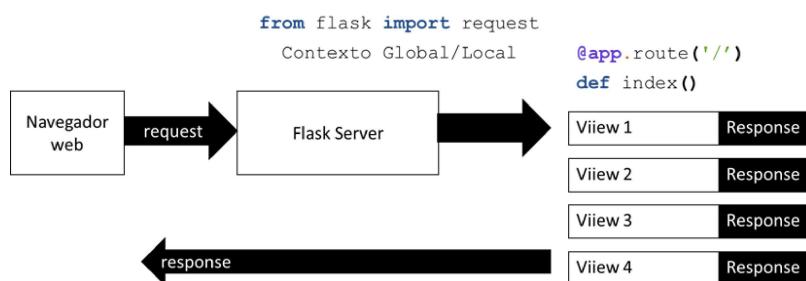


Fonte: elaborada pelo autor.

Essa estrutura leve e simples do Flask permitiu a criação de aplicações focadas em microsserviços e os chamados *micro-frontends* (arquiteturas focadas na construção de *front-ends* autocontidos).

Na Figura 3, você pode observar o ciclo de requisição e de resposta do Flask. A sua estrutura possui muitas facilidades para roteamento, ou seja, a associação entre uma URL com o código ou função que executará uma lógica e devolverá uma resposta. Nesse caso, o Flask server encaminha a requisição para o tratamento adequado que, por sua vez, construirá a resposta necessária. Para renderizar as páginas, Flask disponibiliza um recurso chamado Jinja2, que associa os *templates* usados para a construção desses retornos.

Figura 3 | Ciclo request e response do Flask



Fonte: elaborada pelo autor.

Para comunicação com a camada de banco de dados, pode-se utilizar a extensão Flask para o SQLAlchemy. Dessa forma, é possível também fazer o mapeamento objeto-relacional usando uma biblioteca amplamente utilizada com a estrutura do Flask.

Diante disso, Flask tem como vantagens a velocidade de processamento das requisições e uma complexidade mínima. Apesar de não ter um mecanismo nativo de ORM, permite a utilização de extensões com bibliotecas disponíveis e oferece suporte a NoSQL. Por essas razões, ele se tornou um framework prático e conveniente para muitos desenvolvedores.

Chegamos ao final deste bloco, ao longo do qual foi possível ter uma visão geral sobre o Flask. Como explicado, ele é conveniente para aplicações focadas em microsserviços, *micro-frontends* e aplicações que possam usufruir das capacidades de roteamento providos pelo Flask combinadas com as extensões oferecidas.

Bons estudos!

## VIDEOAULA: FLASK

Nesta videoaula, o objetivo é conhecer o Flask, um microframework muito usado na comunidade de desenvolvedores Python. Assim, você poderá entender os motivos pelos quais Flask é considerado um microframework e que recursos ele entrega inicialmente ao desenvolvedor e como usar eventuais extensões para seus projetos.

Videoaula: Flask

Para visualizar o objeto, acesse seu material digital.

## ESTUDO DE CASO

Você trabalha em uma equipe que desenvolveu um microframework para Python e agora deseja publicar o trabalho para que outros desenvolvedores possam utilizar.

Você já utilizou o pip (*package installer for python*) e tem um repositório chamado PyPI para publicar o trabalho.

Que tal explicar para a equipe como publicar o trabalho no PyPI e indicar o que precisa ser feito? Quais seriam seus passos e como você apresentaria o resultado?

## RESOLUÇÃO DO ESTUDO DE CASO

Diversos times contribuem para projetos de software e, neste estudo de caso, o objetivo foi estimular o seu autoestudo e entender como publicar o seu trabalho para que a comunidade Python possa utilizar.

No caso de Python, muitos projetos estão descritos em PyPI (pronunciado como *Pai-Pee-Eye*) e lá podem ser encontrados frameworks, bibliotecas e outros tipos de componentes úteis. O ponto de partida seria visitar o site <https://pypi.org/>. Ao fazer isso, você poderá ver alguns indicadores, tais como número de projetos catalogados, de *releases*, de arquivos e de usuários. Lá você pode se deparar ainda com um link chamado *Learn how to package your Python code for PyPI*, que pode ser traduzido como “Aprenda como empacotar o seu código Python para PyPI”.

Para isso, você poderá acessar o link <https://packaging.python.org/en/latest/tutorials/packaging-projects/>.

Nesse link, você terá informações sobre como empacotar um projeto a partir da criação dos arquivos do pacote, um diretório de testes, o arquivo `pyproject.toml` com instruções sobre como compilar o projeto. Depois disso devem-se configurar os metadados do projeto classificados como estáticos, que não mudarão ao longo da evolução do projeto, e dinâmicos, quando poderão mudar. Esses dados estarão num arquivo chamado `setup.cfg`.

Outras necessidades serão a criação de um arquivo `README.md`, criação da licença de uso, outros arquivos necessários além da geração dos pacotes para distribuição. Muitas especificações para interoperabilidade de informações são referenciadas como metadados. No Quadro 9, você terá uma visão geral das especificações do PyPA ou *Python Packaging Authority*.

Quadro 9 | Visão das especificações PyPA

	<b>Seção</b>	<b>Exemplo de dados</b>
<b>Package Distribution Metadata</b>	<i>Core metadata specifications</i>	Nome, versão, plataformas suportadas, descrição, link de download, autores, e-mail.
	<i>Version specifiers</i>	Regras para numeração de versões e releases.
	<i>Dependency specifiers</i>	Especificação dos componentes, compatibilidade com versões passadas, copyright.
	<i>Declaring build system dependencies</i>	Informações de dependências relacionadas ao arquivo <code>pyproject.toml</code> .

	<b>Seção</b>	<b>Exemplo de dados</b>
	<i>Declaring project metadata</i>	Informações relacionadas ao arquivo <code>pyproject.toml</code> .
	<i>Platform compatibility tags</i>	Relação com plataformas como Windows, Linux, Mac.
	<i>Recording installed projects</i>	Informações sobre o projeto instalado em determinada plataforma.
	<i>Entry points specification</i>	Pontos de interação com eventuais componentes externos.
	<i>Recording the Direct URL Origin of installed distributions</i>	Integração com Subversion, github e outras ferramentas.

Fonte: elaborado pelo autor.

O principal objetivo deste estudo de caso foi possibilitar o entendimento do que é necessário para construir e distribuir um framework. Apesar de o contexto ser a disponibilização para uma comunidade, muitas das necessidades indicadas também podem ser usadas no contexto de um projeto sendo construído para um cliente que, ao longo do tempo, precisará passar por evolução também.

Que caminhos adicionais você estudou? Como eles se relacionam com o que foi descrito neste estudo de caso?

Bons estudos!

Resolução do Estudo de Caso

Para visualizar o objeto, acesse seu material digital.

## 6º Saiba mais

Nesta aula você compreendeu que o framework Django está baseado em uma arquitetura MVT. Quer saber mais sobre as diferenças entre MVC e MVT?

Então vale a pena ler o texto *The MVT Design Pattern of Django* em <https://python.plainenglish.io/the-mvt-design-pattern-of-django-8fd47c61f582>. Acesso em 11 fev. 2022.

Já sobre o microframework Flask foi mencionado o uso da extensão para o SQLAlchemy. Saiba mais a respeito dessa biblioteca em <https://www.sqlalchemy.org/>. Acesso em 11 fev. 2022.

## REFERÊNCIAS

4 minutos

### Aula 1

GLASSFISH. Eclipse GlassFish. **Eclipse Foundation**, [S. I.], 2021. Disponível em: <https://glassfish.org/>. Acesso em: 6 fev. 2022.

IBM. IBM WebSphere Application Server. **IBM**, [S. I., s. d.]. Disponível em: <https://www.ibm.com/br-pt/cloud/websphere-application-platform>. Acesso em: 6 fev. 2022.

JAKARTA EE. Building an open source ecosystem for cloud native enterprise java. **Jakarta EE**, [S. I., s. d.]. Página inicial. Disponível em: <https://jakarta.ee>. Acesso em: 9 jan. 2022.

JAKARTA ENTERPRISE Beans, Core Features. Version 4.0. Status Final. **Jakarta EE**, [S. /], 5 nov. 2020. Disponível em: <https://jakarta.ee/specifications/enterprise-beans/4.0/jakarta-enterprise-beans-spec-core-4.0.html>. Acesso em: 9 jan. 2022.

JAKARTA PERSISTENCE. Version 3.0. Status Final Release. **Jakarta EE**, [S. /], 8 set. 2020. Disponível em: <https://jakarta.ee/specifications/persistence/3.0/jakarta-persistence-spec-3.0.html>. Acesso em: 11 jan. 2022.

JAKARTA SERVER FACES Specification. Version 3.0. Status Final. **Jakarta EE**, [S. /], 23 set. 2020. Disponível em: <https://jakarta.ee/specifications/faces/3.0/jakarta-faces-3.0.html> Acesso em: 09 de jan. de 2022.

JAKARTA SERVER PAGES Specification. Version 3.0. Status Final. **Jakarta EE**, [S. /], 21 out. 2020. Disponível em: <https://jakarta.ee/specifications/pages/3.0/jakarta-server-pages-spec-3.0.html>. Acesso em: 9 jan. 2022.

JAKARTA SERVLET Specification. Version 5.0. Status Final. **Jakarta EE**, [S. /], 7 set. 2020. Disponível em: <https://jakarta.ee/specifications/servlet/5.0/jakarta-servlet-spec-5.0.html>. Acesso em: 9 jan. 2022.

JBOSS. **JBoss**, [S. /], 2022. Disponível em: <https://www.jboss.org/>. Acesso em: 6 fev. 2022.

ORACLE. Oracle WebLogic Server. **Oracle**, [S. /, s. d.]. Disponível em: <https://www.oracle.com/br/java/weblogic/>. Acesso em: 6 fev. 2022.

SPECIFICATIONS. **Jakarta EE**, [S. /, s. d.]. Disponível em: <https://jakarta.ee/specifications/>. Acesso em: 8 jan. 2022.

THE JAKARTA EE tutorial. Release 9.1. Status Final. **Eclipse EE**, [S. /], dez. 2021. Disponível em: <https://eclipse-ee4j.github.io/jakartaeetutorial/>. Acesso em: 9 jan. 2022.

TIJMS, A. Transition from Java EE to Jakarta EE. **Java Magazine**, [S. /], 27 fev. 2020. Disponível em: <https://blogs.oracle.com/avamagazine/post/transition-from-java-ee-to-jakarta-ee>. Acesso em: 8 jan. 2022.

## Aula 2

ERMIGOTTI, L. 17 JavaScript Frameworks that You Should Know in 2021 – A Comprehensive Guide.

**Codemotion Magazine**, Roma, 2021. Disponível em: <https://www.codemotion.com/magazine/frontend/javascript/javascript-frameworks-guide/>. Acesso em: 17 jan. 2022.

GETTING started with Angular. **Angular**, [S. /], 2021. Disponível em: <https://angular.io/start>. Acesso em: 11 jan. 2022.

INTRODUCTION to Node.js. **Node.js**, [S. /] [2020?]. Disponível em: <https://nodejs.dev/learn>. Acesso em: 10 jan. 2022.

MCLAUGHLIN, B. **Head Rush Ajax**. Massachusetts: O'Reilly, 2006.

NODEJS. **Node.js**, [S. /, s. d.]. Disponível em: <https://nodejs.org/>. Acesso em: 7 fev. 2022.

W3SCHOOLS. AJAX – The XMLHttpRequest Object. **W3 Schools**, [S. /], c2022. Disponível em: [https://www.w3schools.com/xml/ajax\\_xmlhttprequest\\_create.asp](https://www.w3schools.com/xml/ajax_xmlhttprequest_create.asp). Acesso em: 17 jan. 2022.

## Aula 3

BUDIMAN, E. et al. Eloquent Object Relational Mapping Models for Biodiversity Information System. In: INTERNATIONAL CONFERENCE ON COMPUTER APPLICATIONS AND INFORMATION PROCESSING TECHNOLOGY, 4., 2017, Bali. **Proceedings** [...]. [S. /]: IEEE, 2017.

CODEIGNITER. Static Pages. **Codeigniter4**, [S. /], 2022. Disponível em: [https://codeigniter.com/user\\_guide/tutorial/static\\_pages.html](https://codeigniter.com/user_guide/tutorial/static_pages.html). Acesso em: 16 jan. 2022.

CONTROLLERS. **Laravel**, [S. /, s. d.]. Disponível em: <https://laravel.com/docs/8.x/controllers#basic-controllers>. Acesso em: 7 fev. 2022.

CREATE your First Page in Symfony. **Symfony**, [S. /], 2022. Disponível em: [https://symfony.com/doc/current/page\\_creation.html](https://symfony.com/doc/current/page_creation.html). Acesso em: 7 fev. 2022.

DATABASES and the Doctrine ORM. **Symfony**, [S. /], 2022. Disponível em: <https://symfony.com/doc/current/doctrine.html>. Acesso em: 7 fev. 2022.

DOCKER. **Docker**, [S. /, s. d.]. Disponível em: <https://www.docker.com/>. Acesso em: 7 fev. 2022.

ELOQUENT: Getting started. **Laravel**, [S. I., s. d.]. Disponível em: <https://laravel.com/docs/8.x/eloquent>. Acesso em: 7 fev. 2022.

INTRODUCTION. **Symfony**, [S. I.], 2022. Disponível em: [https://symfony.com/doc/current/create\\_framework/introduction.html](https://symfony.com/doc/current/create_framework/introduction.html). Acesso em: 7 fev. 2022.

KUBERNETES. **Kubernetes**, [S. I.], c2022. Disponível em: <https://kubernetes.io/pt-br/>. Acesso em: 7 fev. 2022.

LAMINAS. Laminas Documentation. **Laminas**, [S. I.], c2020. Disponível em: <https://docs.laminas.dev/>. Acesso em: 7 fev. 2022.

LAMINAS. Laminas Project, the enterprise-ready PHP Framework and components. **Laminas**, [S. I., s. d.]. Disponível em: <https://getlaminas.org/>. Acesso em: 20 jan. 2022.

LARAVEL. **Symfony**, [S. I., s. d.]. Disponível em: <https://symfony.com/projects/laravel>. Acesso em: 7 fev. 2022.

LARAVEL. The PHP Framework for Web Artisans. **Laravel**, [S. I.], c2022. Disponível em: <https://laravel.com/>. Acesso em: 20 jan. 2022.

LISBOA, F. **Do PHP ao Laminas** – Domine as boas práticas. São Paulo: Casa do Código, 2020. ISBN 978-65-86110-49-4.

MEU SITE. **Meu site.com**, [S. I., s. d.]. Disponível em: <http://www.meusite.com/hello>. Acesso em: 7 fev. 2022.

PROJECTS using Symfony. **Symfony**, [S. I., s. d.]. Disponível em: <https://symfony.com/projects>. Acesso em: 8 fev. 2022.

SYMFONY. **Symfony**, [S. I.], 2022. Disponível em: <https://symfony.com/>. Acesso em: 20 jan. 2022.

VIEWS. **Laravel**, [S. I., s. d.]. Disponível em: <https://laravel.com/docs/8.x/views>. Acesso em: 7 fev. 2022.

#### Aula 4

AIOHTTP 3.8.1. **PyPI**, [S. I.], 14 nov. 2021. Disponível em: <https://pypi.org/project/aiohttp/>. Acesso em: 8 fev. 2022.

ASYNCIO – Asynchronous I/O. **Python**, [S. I., s. d.]. Disponível em: <https://docs.python.org/3/library/asyncio.html>. Acesso em: 8 fev. 2022.

BIOCO, J., ROCHA, A. Web Application for Management of Scientific Conferences. In: WORLD CONFERENCE ON INFORMATION SYSTEMS AND TECHNOLOGIES, 7. 2019, Ilha da Toxa. **Proceedings** [...]. Suíça: Springer, 2019.

BOTTLE 0.12.19. **PyPI**, [S. I.], 11 nov. 2020. Disponível em: <https://pypi.org/project/bottle/>. Acesso em: 8 fev. 2022.

CHERRYPY 18.6.1. **PyPI**, [S. I.], 3 jul. 2021. Disponível em: <https://pypi.org/project/CherryPy/>. Acesso em: 8 fev. 2022.

CUBICWEB 3.35.0. **PyPI**, [S. I.], 2 fev. 2022. Disponível em: <https://pypi.org/project/cubicweb/>. Acesso em: 8 fev. 2022.

DASH 2.1.0. **PyPI**, [S. I.], 28 jan. 2022. Disponível em: <https://pypi.org/project/dash/>. Acesso em: 8 fev. 2022.

DJANGO. **Django**: The web framework for perfectionists with deadline, [S. I.], c2022. Disponível em: <https://www.djangoproject.com/>. Acesso em: 22 jan. 2022.

DJANGO 4.0.2. **PyPI**, [S. I.], 1 fev. 2022. Disponível em: <https://pypi.org/project/Django/>. Acesso em: 8 fev. 2022.

FALCON 3.0.1. **PyPI**, [S. I.], 10 mar. 2021. Disponível em: <https://pypi.org/project/falcon/>. Acesso em: 8 fev. 2022.

FLASK. Flask web development, one drop at a time. c2022. Disponível em: <https://palletsprojects.com/p/flask/>. Acesso em: 22 jan. 2022.

FLASK 2.0.2. **PyPI**, [S. I.], 4 out. 2021. Disponível em: <https://pypi.org/project/Flask/>. Acesso em: 8 fev. 2022.

GIOTTO 0.10.5. **PyPI**, [S. I.], 18 jan. 2013. Disponível em: <https://pypi.org/project/giotto/>. Acesso em: 8 fev. 2022.

GROWLER 0.8.0. **PyPI**, [S. I.], 16 out. 2016. Disponível em: <https://pypi.org/project/growler/>. Acesso em: 8 fev. 2022.

HUG 2.6.1. **PyPI**, [S. I.], 6 fev. 2020. Disponível em: <https://pypi.org/project/hug/>. Acesso em: 8 fev. 2022.

MOREPATH 0.19. **PyPI**, [S. /], 29 jan. 2020. Disponível em: <https://pypi.org/project/morepath/>. Acesso em: 8 fev. 2022.

PACKAGING Python Projects. **PyPA**, [S. /], 23 jan. 2022. Disponível em: <https://packaging.python.org/en/latest/tutorials/packaging-projects/>. Acesso em: 8 fev. 2022.

PYCNIC 0.1.8. **PyPI**, [S. /], 29 jan. 2020. Disponível em: <https://pypi.org/project/pycnic/>. Acesso em: 8 fev. 2022.

PYLONS 1.0.3. **PyPI**, [S. /], 12 jan. 2018. Disponível em: <https://pypi.org/project/Pylons/>. Acesso em: 8 fev. 2022.

PYPI. Find, install and publish Python packages with the Python Package Index. **Python Package Index (PyPI)**, [S. /], c2022. Disponível em: <https://pypi.org/>. Acesso em: 22 jan. 2022.

PYRAMID 2.0. **PyPI**, [S. /], 1 mar. 2021. Disponível em: <https://pypi.org/project/pyramid/>. Acesso em: 8 fev. 2022.

PYTHON. **Python**, [S. /], c2022. <https://www.python.org/>. Acesso em: 22 jan. 2022.

SANIC 21.12.1. **PyPI**, [S. /], 6 jan. 2022. Disponível em: <https://pypi.org/project/sanic/>. Acesso em: 8 fev. 2022.

SHADHIN, F. The MVT Design Pattern of Django – Understand the Model-View-Template architecture of a Django application. **Python**, [S. /], 4 maio 2021. Disponível em: <https://python.plainenglish.io/the-mvt-design-pattern-of-django-8fd47c61f582>. Acesso em: 8 fev. 2022.

SINGH, V. Best Python Frameworks. **Hackr.io**, [S. /], 7 jan. 2022. Disponível em: <https://hackr.io/blog/python-frameworks>. Acesso em: 28 jan. 2022.

SQLALCHEMY. **SQLAlchemy**, [S. /, s. d.]. Disponível em: <https://www.sqlalchemy.org/>. Acesso em: 8 fev. 2022.

SQLALCHEMY in Flask. **Flask**, [S. /], c2010. Disponível em: <https://flask.palletsprojects.com/en/2.0.x/patterns/sqlalchemy/>. Acesso em: 28 jan. 2022.

TORNADO 6.1. **PyPI**, [S. /], 30 out. 2020. Disponível em: <https://pypi.org/project/tornado/>. Acesso em: 8 fev. 2022.

TURBOGEARS 1.5.1. **PyPI**, [S. /], 26 nov. 2011. Disponível em: <https://pypi.org/project/TurboGears/>. Acesso em: 8 fev. 2022.

WEB2PY 2.1.1. **PyPI**, [S. /], 26 nov. 2011. Disponível em: <https://pypi.org/project/web2py/>. Acesso em: 8 fev. 2022.