

COM222

DESENVOLVIMENTO DE

SISTEMAS WEB

Aula 10: Introdução à Programação
server-side

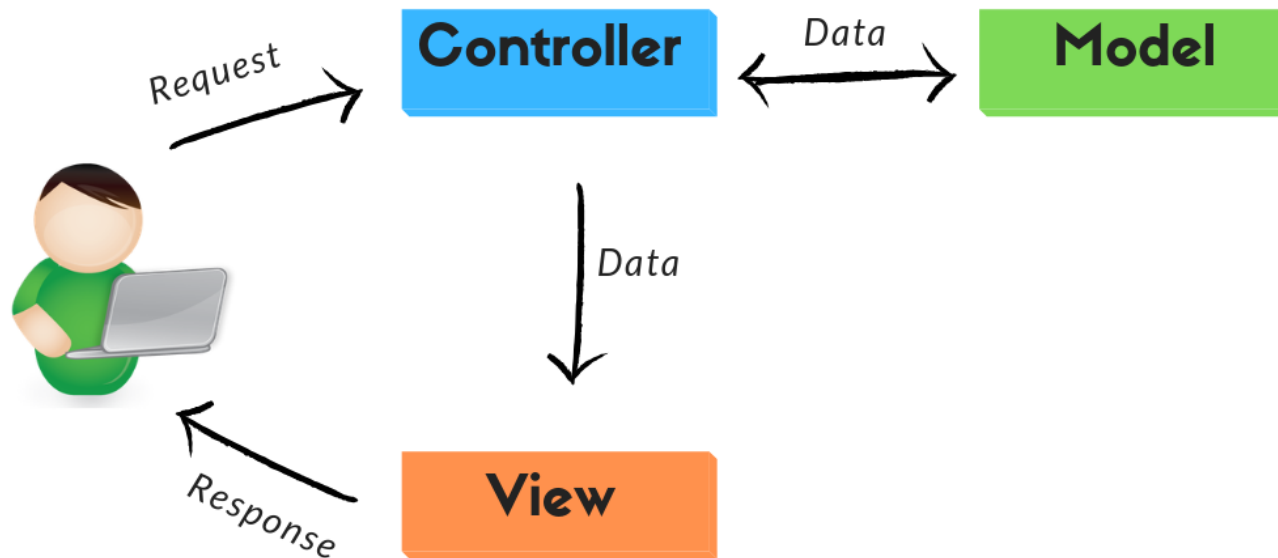
Conteúdo

- ❑ Arquitetura MVC
- ❑ Programação server-side com Node.js e Express

Arquitetura MVC

3

- ❑ Objetivo de projeto: identificar quais componentes fazem parte do modelo, da visão e do controle da aplicação



Usuário não pode modificar diretamente o modelo

Implementação do MVC

4

- ❑ Baseada no conceito de acoplamento fraco
- ❑ A implementação das 3 camadas deve estar separada
- ❑ Objetos de uma camada não dependem da forma como objetos das outras camadas são implementados
- ❑ A aplicação resultante
 - ▣ Fica mais fácil de construir e manter
 - ▣ É implementada mais rapidamente
 - ▣ Fica mais **robusta** (resiliente a erros de execução)

MVC – Modelo

- Representa a **informação persistente** mantida pela aplicação
- Essa informação pode ser mantida em um banco de dados como o MongoDB
 - ▣ que vamos cobrir mais a frente (MEAN)

MVC – Visão

6

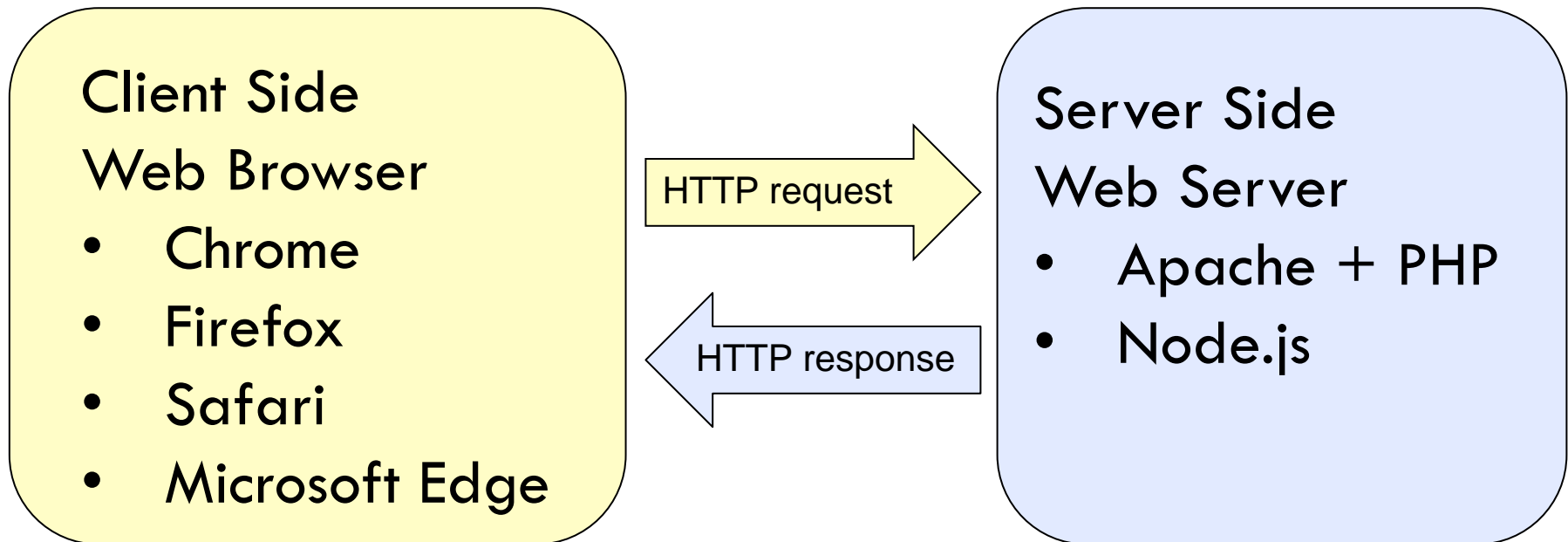
- ❑ Objetos da visão estão associados à **interface de usuário** da aplicação
 - ❑ Componentes de interação de uma página web
- ❑ Em cada caso de uso, usuários interagem com pelo menos um componente de visão
- ❑ Componentes de visão coletam informação do usuário para ser usada pelos componentes de controle e de modelo

MVC – Controle

7

- Coordena os componentes de modelo e de visão
 - ▣ Em geral não tem um corresponde físico no mundo real
 - ▣ Coleta informação na camada de visão e a disponibiliza para o modelo
 - É dessa forma que as informações coletadas dos usuários atualizam o modelo
- Representa fluxos de controle da aplicação
 - ▣ Implementado através de roteamento

Aplicações web cliente-servidor



- HTTP request
 - ▣ GET / POST
- HTTP response
 - ▣ Página HTML gerada dinamicamente

Rotas

9

- Uma **rota** é um mapeamento entre uma requisição HTTP do cliente e o controlador apropriado no servidor
 - ▣ Associa uma **URI** e um **método HTTP** a uma ação particular do **controlador**
- Uma **URI** (Uniform Resource Identifier) identifica um recurso que a aplicação web provê
 - ▣ Exemplo de recurso: uma página web
 - ▣ A forma mais comum de **URI** é a **URL**

Métodos HTTP

10

- Uma **rota** é um mapeamento entre uma requisição HTTP do cliente e o controlador apropriado no servidor
 - ▣ Associa uma **URI** e um **método HTTP** a uma ação particular do **controlador**
- Um **método HTTP** é também chamado de **verbo HTTP**
- Verbos HTTP mais usados são **GET**, **POST**, **PUT** e **DELETE**
 - ▣ Definidos pelo protocolo HTTP

Servindo páginas web

11

- ❑ Servidor web serve páginas web
 - ▣ Exibidas nos browsers
 - ▣ Podem ser **estáticas** ou **dinâmicas**
- ❑ Páginas web estáticas: **.html**
 - ▣ Arquivos HTML armazenados no servidor
 - ▣ Uma página estática sempre mostra o mesmo conteúdo
- ❑ Páginas web dinâmicas: **.js**
 - ▣ Geradas através de código Javascript no servidor
 - ▣ Contém conteúdo dinâmico

node.js

12

- ❑ Neste curso usaremos o servidor **node.js**
 - ❑ nodejs.org
 - ❑ Servidor rápido e leve
 - ❑ Criado em 2009 como uma alternativa ao servidor web Apache
 - ❑ Codificado em **JavaScript**

node.js

13

- ❑ Usa a engine open source V8 do Google Chrome
 - ▣ Just-in-time compilation, automatic inlining, dynamic code optimization, etc.
- ❑ Conexões são atendidas num loop de **única thread**
 - ▣ Threads separadas funcionam como background workers
- ❑ Cada conexão invoca uma **função de callback** Javascript
 - ▣ Utiliza threads a partir de um pool de threads
 - ▣ Manipula I/O sem bloqueio

node.js e Express

14

- ❑ Instalar node.js: <https://nodejs.org/>
- ❑ node.js utiliza pacotes
 - ❑ Instalar pacotes com **npm**, **node package manager**.
- ❑ Nós vamos usar o framework Express
 - ❑ Suporta a arquitetura MVC
 - ❑ Comando de instalação:

```
npm install -S express -g
```

Interação via forms

15

- ❑ Formulários são a forma mais comum de interação entre um cliente e uma aplicação web
- ❑ Quando o botão de Submit é pressionado, o navegador envia os dados do formulário para programa Javascript no servidor
- ❑ O programa Javascript pode usar os dados do form para
 - ▣ Consultar um banco de dados no back-end
 - ▣ Gerar uma página web e enviá-la para o cliente

Formas de enviar dados

16

- ❑ O atributo action da tag <FORM> indica para onde os dados do formulário devem ser enviados
- ❑ Dois métodos podem ser usados para enviar dados para o servidor
 - ❑ GET
 - ❑ POST

Formas de enviar dados

17

□ GET

- ▣ Os dados são inseridos na própria URL
- ▣ Bom para pequenos volumes de dados
- ▣ Os dados ficam visíveis. Exemplo:

```
http://localhost:8080/response?nome=Laercio&sobrenome=Baldochi
```

□ POST

- ▣ Os dados são enviados por meio de variáveis de ambiente
- ▣ Bom para volumes maiores de informação
- ▣ Ligeiramente mais seguro que o método GET

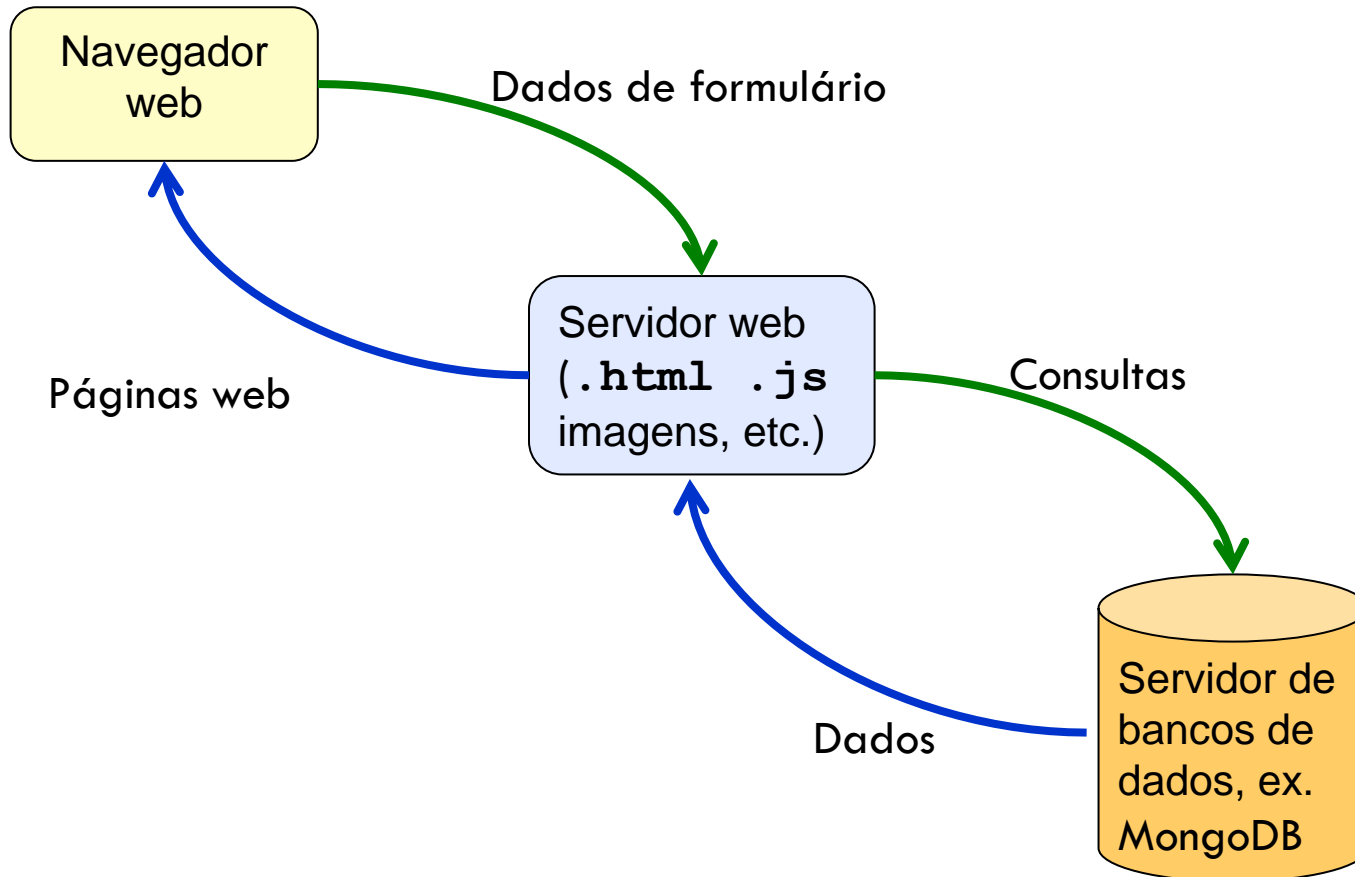
Sugestão para uso de GET e POST

18

- ❑ GET deve ser usado para requisitar páginas/forms de um servidor
- ❑ POST deve ser usado para enviar dados de formulário para o servidor

Arquitetura web de 3 camadas

19



Aplicação Hello World

20

- ❑ Vamos criar uma aplicação Hello World utilizando o Express
 - ▣ “O Express é um framework para aplicativo da web do Node.js mínimo e flexível que fornece um conjunto robusto de recursos para aplicativos web e móvel.”
 - ▣ Facilita a criação de aplicações server-side e APIs
- ❑ Vamos começar instalando o Express
 - ▣ `npm install -g express-generator`
- ❑ Em seguida podemos criar nossa aplicação
 - ▣ `express -e helloworld`

Aplicação Hello World

21

- ❑ `express -e helloworld`
 - ▣ Opção `-e` indica que vamos usar a engine de renderização EJS (o padrão é Jade/PUG)
- ❑ EJS = Embedded JavaScript templates
 - ▣ Engine básica
 - ▣ Faz binding de variáveis entre rotas (controllers) e views

Aplicação Hello World

22

- ❑ Após a execução do comando `express`, temos a geração de uma aplicação server-side com a seguinte estrutura

```
.
├── app.js
├── bin
│   └── www
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.pug
    ├── index.pug
    └── layout.pug
```

Aplicação Hello World

23

```
.  
├─ app.js  
├─ bin  
│   └─ www  
├─ package.json  
├─ public  
│   ├── images  
│   ├── javascripts  
│   └── stylecss  
│       └─ style.css  
├─ routes  
│   ├── index.js  
│   └─ users.js  
└─ views  
    ├── error.pug  
    ├── index.pug  
    └─ layout.pug
```

www

Responsável por iniciar o
servidor web do Express

Aplicação Hello World

24

```
.
├── app.js
├── bin
│   └── www
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.pug
    ├── index.pug
    └── layout.pug
```

app.js

Responsável por configurar a aplicação web

- Importação de pacotes
- Criação de variáveis
- Definição de arquivos de rotas e views

Aplicação Hello World

25

```
.
├── app.js
├── bin
│   └── www
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.pug
    ├── index.pug
    └── layout.pug
```

public

Mantém os recursos públicos usados pela aplicação:
folhas de estilo CSS, arquivos Js, imagens

Aplicação Hello World

26

```
.
├── app.js
├── bin
│   └── www
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.pug
    ├── index.pug
    └── layout.pug
```

routes

Para cada rota definida em app.js é necessário definir um arquivo na pasta routes

Rotas são controllers

Aplicação Hello World

27

```
.  
├── app.js  
├── bin  
│   └── www  
├── package.json  
├── public  
│   ├── images  
│   ├── javascripts  
│   └── stylesheets  
│       └── style.css  
├── routes  
│   ├── index.js  
│   └── users.js  
└── views  
    ├── error.pug  
    ├── index.pug  
    └── layout.pug
```

views

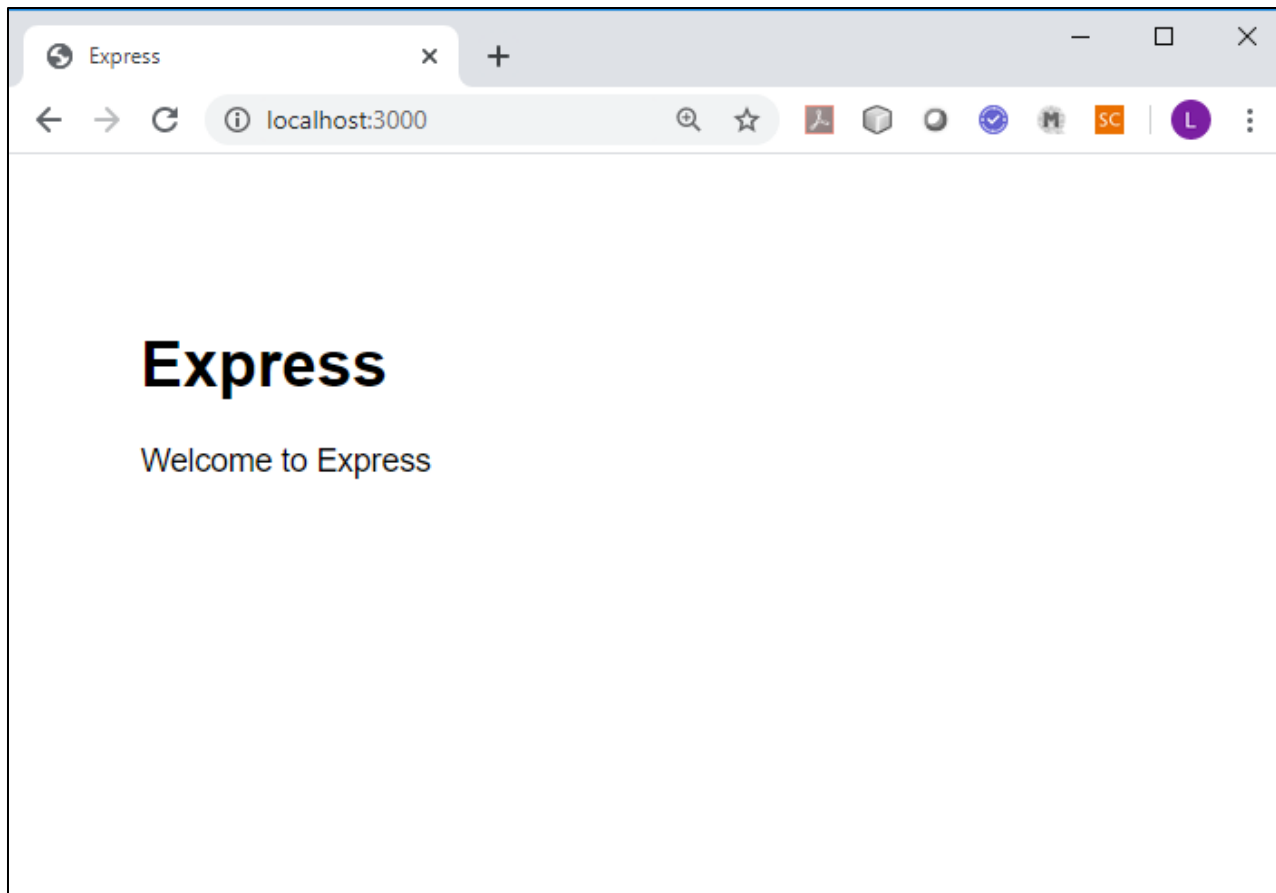
Em geral, para cada rota deve haver uma view

Ao invés de Jade/PUG, vamos usar EJS

Aplicação Hello World

28

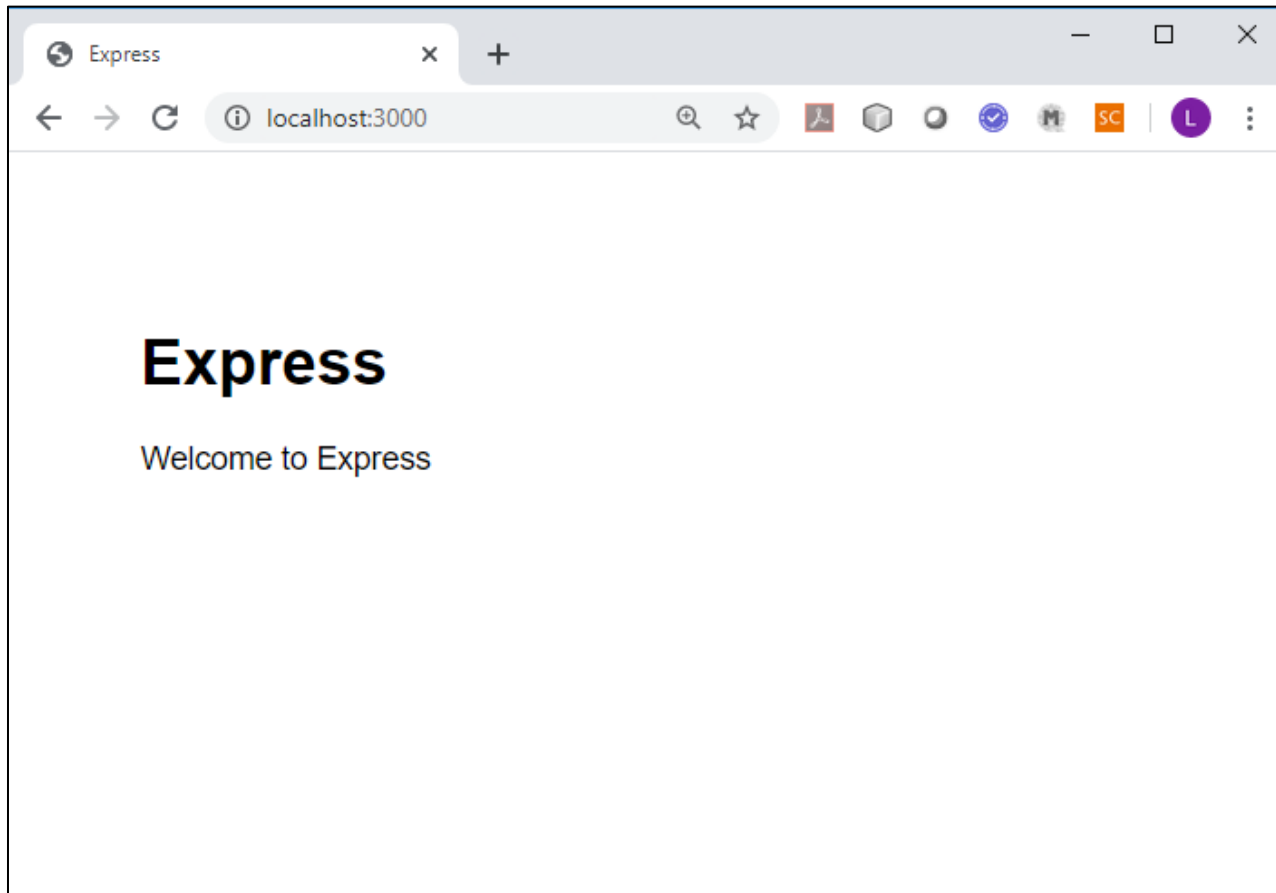
- ❑ Template default do Express aparece assim:



Aplicação Hello World

29

- ❑ Para aparecer Hello World! temos que alterar a variável `title` e depois mudar a view

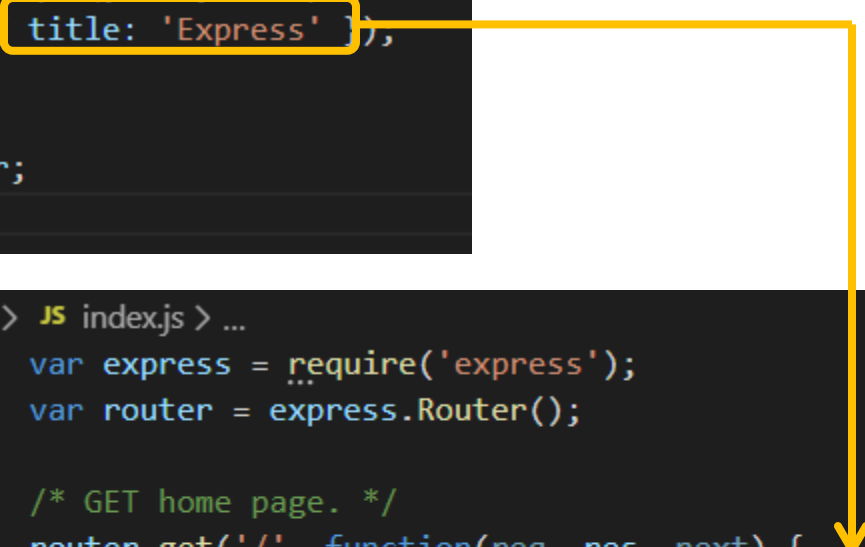


Aplicação Hello World

30

```
routes > JS index.js > ...
1  var express = require('express');
2  var router = express.Router();
3
4  /* GET home page. */
5  router.get('/', function(req, res, next) {
6    res.render('index', { title: 'Express' });
7  });
8
9  module.exports = router;
10
```

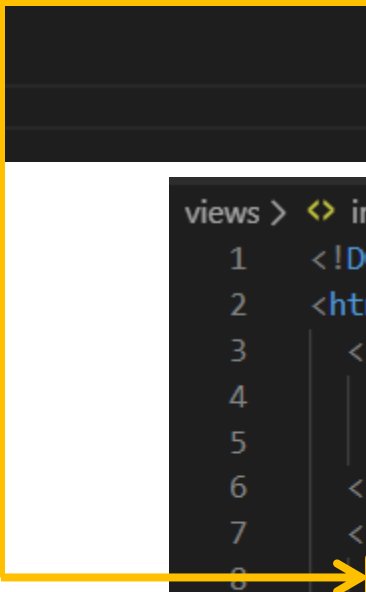
```
routes > JS index.js > ...
1  var express = require('express');
2  var router = express.Router();
3
4  /* GET home page. */
5  router.get('/', function(req, res, next) {
6    res.render('index', { mensagem: 'Hello World!' });
7  });
8
9  module.exports = router;
```



Aplicação Hello World

```
views > <> index.ejs > ...
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title><%= title %></title>
5      <link rel='stylesheet' href='/stylesheets/style.css' />
6    </head>
7    <body>
8      <h1><%= title %></h1>
9      <p>Welcome to <%= title %></p>
10   </body>
11 </html>
12
```

```
views > <> index.ejs > ...
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title><%= title %></title>
5      <link rel='stylesheet' href='/stylesheets/style.css' />
6    </head>
7    <body>
8      <h1><%= mensagem %></h1>
9    </body>
10 </html>
11
```



Exercício 1

32

- ❑ Crie a rota /ola
 - ▣ Exiba: “Olá mundo!”

- ❑ Crie a rota /adeus
 - ▣ Exiba: “Adeus mundo cruel!”

Manipulação de forms

33

- ❑ Vamos agora criar uma aplicação Express para fazer manipulação de forms
- ❑ Neste exemplo usaremos uma template engine mais robusta, chamada Handlebars (hbs)

```
>express -v hbs registra-usr  
>cd registra-usr  
>npm install  
>npm install hbs@4.1.0
```

■ Substitua o conteúdo de *views/layout.hbs*

```
<!doctype html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <meta name="description" content="">
  <meta name="author" content="">

  <title>{{title}}</title>

  <!-- Bootstrap core CSS -->
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
    integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
    crossorigin="anonymous">

  <!-- Custom styles for this template -->
  <link href="/stylesheets/style.css" rel="stylesheet">
</head>

<body>
  {{{body}}}
</body>
</html>
```

❑ Acrescente código a seguir em *public/stylesheets/style.css*

```
html,
body {
  height: 100%;}

body {
  display: -ms-flexbox;
  display: flex;
  -ms-flex-align: center;
  align-items: center;
  padding-top: 40px;
  padding-bottom: 40px;
  background-color: #f5f5f5;}

.full-width {
  width: 100%;
  padding: 15px;
  margin: auto;}

.form-signin {
  max-width: 330px; }

.form-signin .checkbox {
  font-weight: 400; }
```

```
.form-signin .form-control {
  position: relative;
  box-sizing: border-box;
  height: auto;
  padding: 10px;
  font-size: 16px; }

.form-signin .form-control:focus {
  z-index: 2; }

.form-signin input {
  border-radius: 0;
  margin-bottom: -1px; }

.form-signin input:first-of-type {
  border-top-left-radius: 0.25rem;
  border-top-right-radius: 0.25rem; }

.form-signin input:last-of-type {
  border-bottom-left-radius: 0.25rem;
  border-bottom-right-radius: 0.25rem;
  margin-bottom: 10px;
}
```

- ❑ Substitua o conteúdo de *views/index.hbs*
 - ▣ Essa será nossa página home
 - ▣ Cria um form que faz um POST para a rota /registra

```
<form action="/registra" method="POST" class="form-signin full-width text-center">
  <h1 class="h3 mb-3 font-weight-normal">Participe da nossa mailing list</h1>
  <label for="name" class="sr-only">Nome</label>
  <input type="text" name="name" class="form-control" placeholder="Seu nome" required autofocus>
  <label for="email" class="sr-only">Email</label>
  <input type="email" name="email" class="form-control" placeholder="Seu email" required>
  <label for="emailConfirmation" class="sr-only">Email (confirmação)</label>
  <input type="email" name="emailConfirmation" class="form-
control" placeholder="Seu email (confirmação)" required>
  <button class="btn btn-lg btn-primary btn-block" type="submit">Registra</button>
  <p class="mt-5 mb-3 text-muted">&copy; 2020</p>
</form>
```

- ❑ Acrescente o código a seguir em *routes/index.js*
 - ▣ Acrescentar após a rota '/'
- ❑ **name** e **email** são obtidos a partir de req.body

```
router.post("/registra", function(req, res, next) {  
  const { name, email } = req.body;  
  
  // Numa aplicação real: validar usuário; registrá-lo no banco; enviar email de confirmação  
  
  res.render("registrado", {  
    title: "Você está registrado",  
    name,  
    email  
  });  
});
```

- ❑ Por fim, basta criar a página [views/registrado.hbs](#)
 - ❑ name e email que foram ajustados na rota agora são usados no template

```
<div class="full-width text-center">
  <h1 class="display-3">Obrigado, {{name}}!</h1>
  <p class="lead"><strong>Por favor, verifique seu email</strong> para confirmar seu registro
em nossa newsletter.</p>
  <hr>
  <p>
    Algum problema? <a href="">Contacte-nos</a>
  </p>
  <p class="lead">
    <a class="btn btn-primary btn-sm" href="/" role="button">Voltar</a>
  </p>
</div>
```

- ❑ Digite `npm start`
 - ❑ Verifique o resultado em `http://localhost:3000`

Exercício 2

- ❑ Considere o formulário a seguir:



The image shows a web browser window with the title 'Exercício Form'. The address bar displays 'localhost:8080/javascript/exerc'. The page content is a registration form with a light blue background. The form has a purple header with the title 'Registro de usuário'. Below the header, there are two input fields for 'Nome' and 'Idade'. Below these is a single input field for 'E-Mail'. Then, there is a section titled 'Em quais linguagens você programa?' with six checkboxes for 'Java', 'C++', 'C', 'Perl', 'COBOL', and 'VB'. Below this is another section titled 'Com qual frequência poderemos notificá-lo sobre novidades?' with three radio buttons for 'Semanalmente', 'Mensalmente', and 'Quinzenalmente'. At the bottom of the form is a 'Submit' button.

Registro de usuário

Nome

Idade

E-Mail

Em quais linguagens você programa?

☐ Java ☐ C++ ☐ C
☐ Perl ☐ COBOL ☐ VB

Com qual frequência poderemos notificá-lo sobre novidades?

☒ Semanalmente ☐ Mensalmente ☐ Quinzenalmente

Exercício 2

40

- ❑ Crie uma aplicação utilizando Express que exiba o formulário do slide anterior
- ❑ Após o pressionamento do botão Submit, a aplicação deve mostrar uma página de resposta que exiba as informações preenchidas:
 - ❑ Nome idade e-mail
 - ❑ Relação das linguagens selecionadas
 - ❑ A periodicidade de notificação escolhida
- ❑ Código do formulário está disponível em:
 - ❑ <https://baldochi.unifei.edu.br/COM222/exercicioAula10.txt>