



ARQUITETURA SERVER-SIDE

Cassio Trindade - Aula 01

Professores

CASSIO TRINDADE

Professor Convidado

Profissional da área de TI, trabalhando há mais de uma década com a formação de profissionais, dando aulas no Instituto Federal do Rio Grande do Sul, na Faculdade Dom Bosco, Universidade Luterana do Brasil (ULBRA), Pontifícia Universidade Católica do RS (PUCRS) e na TargetTrust. Atualmente atuando como Arquiteto de Software na PUCRS, sendo responsável pela condução e elaboração de mais de 90 projetos diretamente com alunos do curso de Engenharia de Software, trabalhando com as mais variadas tecnologias. Mais de 30 anos de experiência nas áreas de desenvolvimento de software, aplicativos para celulares e sistemas corporativos e para internet desde projetos de e-commerce para o Sonae Portugal e site de classificados digitais do Grupo RBS a dezenas de aplicativos mobiles.

Ementa da disciplina

Estudo sobre Arquitetura cliente-servidor para aplicações web. Introdução aos frameworks MVC server-side: Node.js, Express, Nestjs. Estudo de programação assíncrona e programação reativa. Desenvolvimento de aplicações web com o conceito de uso de serviços.

A close-up photograph of a person's hand wearing a light-colored cuff-link shirt sleeve, typing on a silver laptop keyboard. The background is a dark, abstract space filled with glowing blue binary digits (0s and 1s) and small white dots, suggesting a digital or server-side environment.

Arquitetura Server-Side

Cássio Trindade

Agenda



**Introdução aos frameworks
MVC server-side**

**Programação assíncrona e
programação reativa**

Node.js e Express,

Nest.js

Agenda



Introdução aos frameworks MVC server-side

Programação assíncrona e
programação reativa

Node.js e Express,

Nest.js

Introdução ao MVC

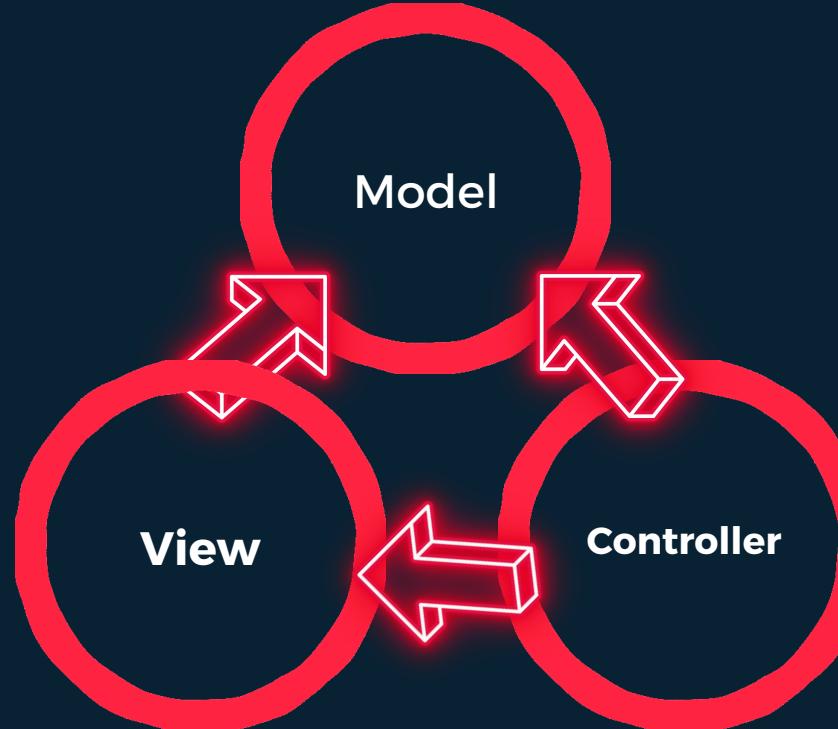
Conceito de arquitetura **Model-View-Controller**

Separação entre dados (Model), apresentação (View) e lógica de negócios (Controller).



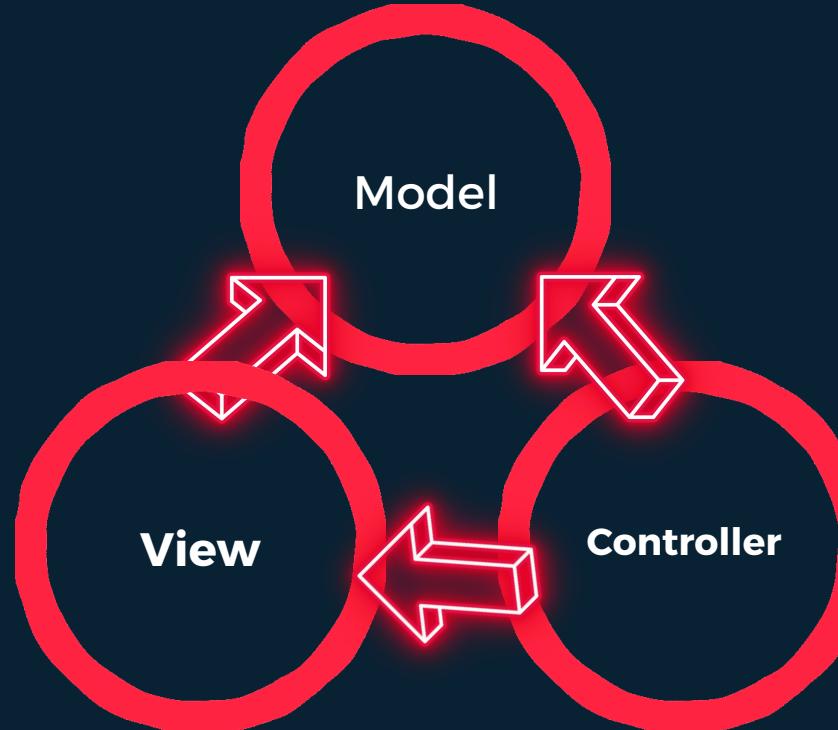
MVC é provavelmente uma das arquiteturas mais populares para aplicativos.
Essa arquitetura foi concebido na linguagem Smalltalk

Model: Esta é a parte do nosso aplicativo que gerencia o banco de dados e todas as operações relacionadas aos dados. Ele cuida do armazenamento, recuperação e manipulação dos dados essenciais para o funcionamento da aplicação.



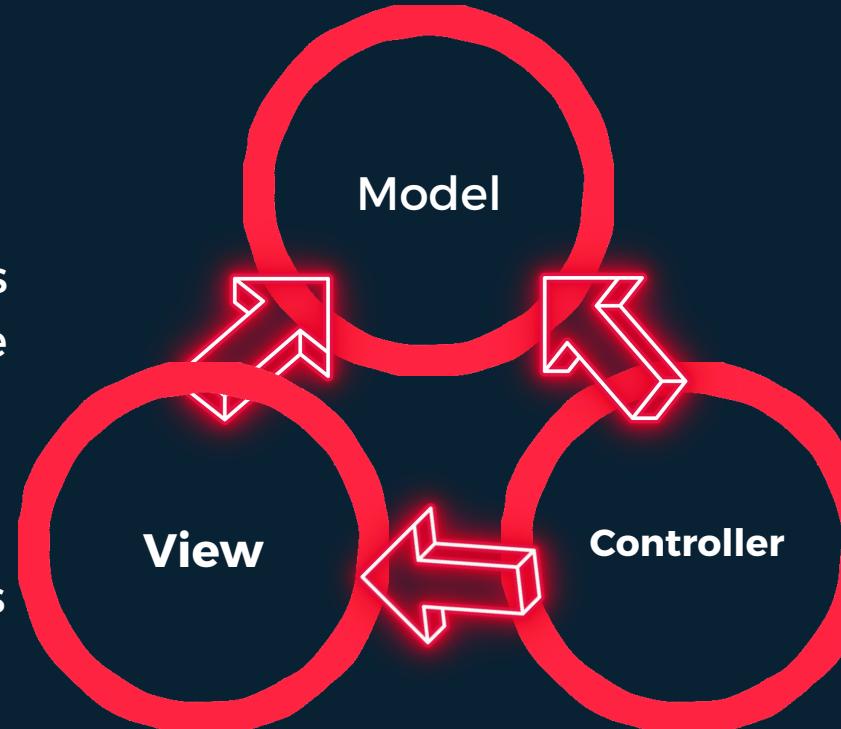
MVC é provavelmente uma das arquiteturas mais populares para aplicativos.
Essa arquitetura foi concebido na linguagem Smalltalk

View: A "Visão" engloba tudo o que é visível para o usuário. Em termos simples, são as páginas e elementos visuais que são apresentados ao cliente. A visão é responsável por garantir uma experiência agradável ao usuário, cuidando da forma como os dados são apresentados.

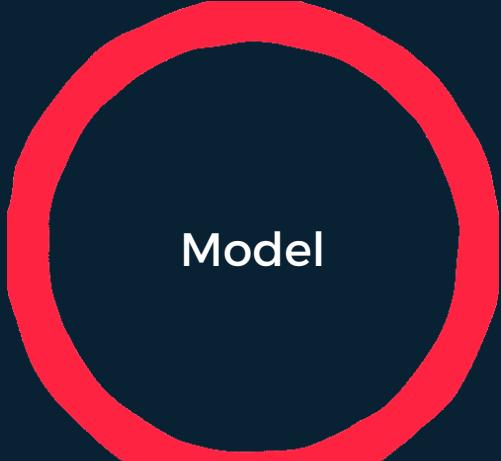


MVC é provavelmente uma das arquiteturas mais populares para aplicativos. Essa arquitetura foi concebido na linguagem Smalltalk

Controller: O "Controlador" é o cérebro da nossa aplicação. Ele contém a lógica que coordena a interação entre o modelo e a visão. No controlador, chamamos os modelos para obter dados, processamos esses dados e os disponibilizamos para as visões para que sejam entregues aos usuários. Além disso, o controlador é o local onde desenvolvemos funcionalidades, as expandimos e realizamos a manutenção da aplicação.



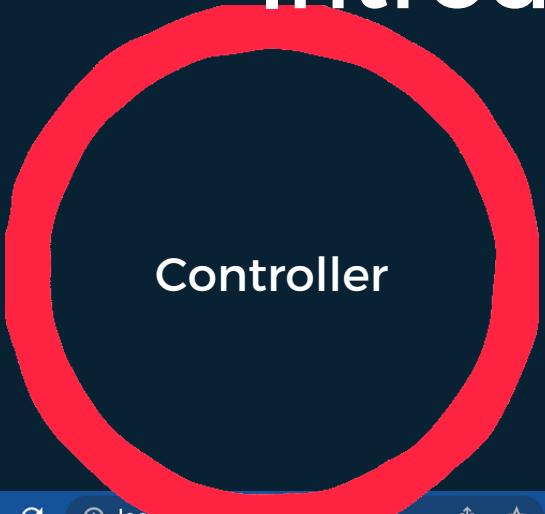
Introdução ao MVC

A screenshot of a web browser window. The address bar shows "localhost:3000/tasks". The main content area displays a heading "Minha Lista de Tarefas" and a list of three tasks:

1. Comprar mantimentos | Pegar leite, ovos e pão no supermercado.
2. Estudar programação | Aprender sobre Node.js e Express.
3. Fazer exercícios | 30 minutos de exercícios aeróbicos.

```
1 const tasks = [
2   {
3     id: 1,
4     title: 'Comprar mantimentos',
5     description: 'Pegar leite, ovos e pão no supermercado.',
6     completed: false,
7     created_at: new Date(),
8   },
9   {
10     id: 2,
11     title: 'Estudar programação',
12     description: 'Aprender sobre Node.js e Express.',
13     completed: false,
14     created_at: new Date(),
15   },
16   {
17     id: 3,
18     title: 'Fazer exercícios',
19     description: '30 minutos de exercícios aeróbicos.',
20     completed: true,
21     created_at: new Date(),
22   }
23 ];
24 module.exports = {
25   getAllTasks: () => tasks,
26   addTask: (task) => tasks.push(task),
27 };
```

Introdução ao MVC



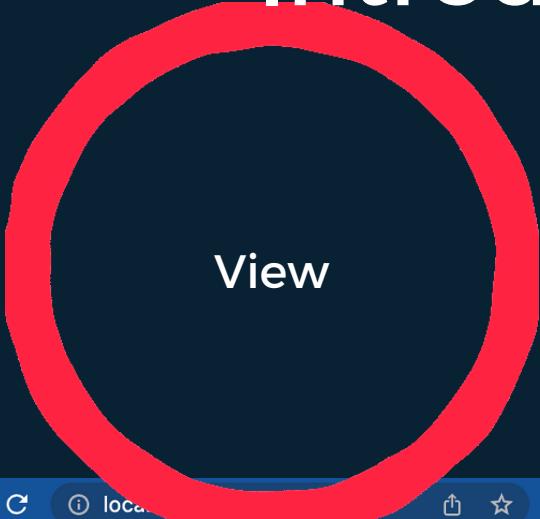
Controller

Minha Lista de Tarefas

1. Comprar mantimentos | Pegar leite, ovos e pão no supermercado.
 2. Estudar programação | Aprender sobre Node.js e Express.
 3. Fazer exercícios | 30 minutos de exercícios aeróbicos.

```
1 const express = require('express');
2 const router = express.Router();
3 const taskModel = require('../models/task');
4
5 router.get('/tasks', (req, res) => {
6   const tasks = taskModel.getAllTasks();
7   res.render('task_list', { tasks });
8 });
9
10 module.exports = router;
```

Introdução ao MVC



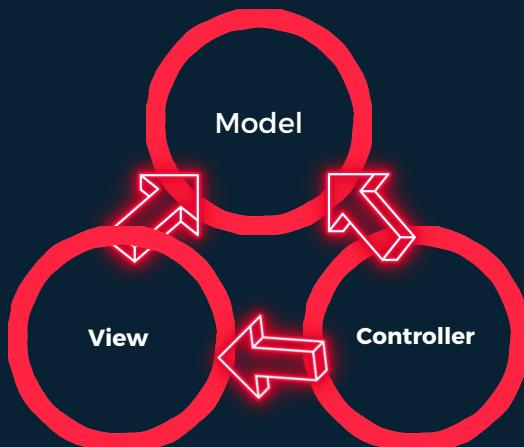
A screenshot of a web browser window. The title bar shows the URL "localhost". The main content area displays a heading "Minha Lista de Tarefas" and a list of three tasks:

- 1. Comprar mantimentos | Pegar leite, ovos e pão no supermercado.
- 2. Estudar programação | Aprender sobre Node.js e Express.
- 3. Fazer exercícios | 30 minutos de exercícios aeróbicos.



```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Lista de Tarefas</title>
5  </head>
6  <body>
7      <h1>Minha Lista de Tarefas</h1>
8      <ol>
9          <% tasks.forEach(task => { %>
10              <li><%= task.title %> | <%= task.description %></li>
11          <% }); %>
12      </ol>
13  </body>
14 </html>
```

Benefícios do MVC



- ✓ Separação de responsabilidade
- ✓ Reutilização de código
- ✓ Facilidade de manutenção
- ✓ Escalabilidade
- ✓ Facilidade de colaboração
- ✓ Facilidade de migração
- ✓ Testabilidade e Documentação

Agenda



**Introdução aos frameworks
MVC server-side**

**Programação assíncrona e
programação reativa**

Node.js e Express,

Nest.js

Programação assíncrona

Paradigma de programação que permite que uma aplicação execute tarefas em paralelo, em vez de sequencialmente, o que é comum em programação síncrona. Isso significa que, em vez de esperar que uma tarefa seja concluída antes de iniciar outra, as tarefas podem ser executadas simultaneamente ou em segundo plano, permitindo que o programa continue sua execução principal sem bloquear.

Programação assíncrona



Pagina
inicial



Menu



Cadastro



Pesquisa



Listagem

Assíncrono



Assíncrona



segundos



Programação assíncrona



```
1 const fs = require('fs');
2
3 // Lê o arquivo de forma assíncrona
4 fs.readFile('exemplo.tx', 'utf8', (err, data) => {
5   if (err) {
6     console.error('Erro ao ler o arquivo:', err);
7     return;
8   }
9   console.log('Conteúdo do arquivo:', data);
10 });
11
12 console.log('Lendo o arquivo...');
```

```
● (base) bash526@exemplo-assincrono\: node app.js
Lendo o arquivo...
Erro ao ler o arquivo: [Error: ENOENT: no such file or directory, open 'exemplo.tx'] {
  errno: -2,
  code: 'ENOENT',
  syscall: 'open',
  path: 'exemplo.tx'
}
● (base) bash527@exemplo-assincrono\: █
```

Programação reativa

Paradigma de programação que se concentra na criação de sistemas e aplicativos que respondem automaticamente a mudanças de estado e eventos. Ela é particularmente útil para desenvolver aplicativos em tempo real, como aplicativos da web que exigem atualizações em tempo real com base em ações do usuário ou em eventos de sistema.



Programação Reativa - A onda do momento!
Hoje em dia muitas empresas estão migrando seus
legados para uma arquitetura de micro-serviços. A
grande preocupação gira em torno da performance ...

Programação reativa

- Observáveis e Observadores (Observable/Observer)
- Bibliotecas de Programação Reativa
- Fluxos de Eventos
- Assinatura e Cancelamento
- Tratamento de Erros
- Aplicações em Tempo Real
- Integração com Bancos de Dados
- Aplicações Assíncronas de Alto Desempenho concorrentes.
- Testes Unitários
- Padrões de Design Reativos
- **Operadores Reativos**

Programação reativa



```
1 const { Observable } = require('rxjs');
2 const { map } = require('rxjs/operators');
3 const numbers = [1, 2, 3, 4, 5];
4
5 const observable = new Observable((observer) => {
6   numbers.forEach((num) => {
7     observer.next(num);
8   });
9   observer.complete();
10 });
11
12 const doubledObservable = observable.pipe(
13   map((value) => value * 2)
14 );
15
16 doubledObservable.subscribe((value) => {
17   console.log(value); // Imprime 2, 4, 6, 8, 10
18 });
19
```

O operador map é usado para transformar os elementos de um fluxo de dados. Vamos criar um exemplo simples que dobra os valores emitidos por um observável:

Agenda



**Introdução aos frameworks
MVC server-side**

**Programação assíncrona e
programação reativa**

Node.js e Express,

Nest.js

Node.js

"..é um interpretador JavaScript do lado do servidor que altera a noção de como um servidor deveria funcionar. Possibilita que um programador crie aplicativos altamente escalável se escreva código que manipule dezenas de milhares de conexões simultâneas.."

Node.js

"..é uma plataforma construída sobre o motor JavaScript do Google Chrome para facilmente construir aplicações de rede rápidas e escaláveis. Node.js usa um modelo de I/O direcionada a evento não bloqueante que o torna leve e eficiente, ideal para aplicações em tempo real com troca intensa de dados através de dispositivos distribuídos.."

Node.js

"..é uma plataforma para desenvolvimento de aplicações server-side baseadas em rede utilizando JavaScript e o V8 Engine, ou seja, com Node.js podemos criar uma variedade de aplicações Web utilizando apenas código em JavaScript.."

Caracteristica Node.js

- Não Bloqueante (Non-Blocking-Thread)
- Javascript Engine V8 - altamente escalável
- Single-thread - cada aplicação terá instância de um único processo.
- Orientado a eventos - a mesma logica a de orientação de eventos do Javascript client-sided

Instalação Node.js

Node.js® is an open-source, cross-platform JavaScript runtime environment.

[Download Node.js®](#)

20.9.0 LTS

Recommended For Most Users

21.1.0 Current

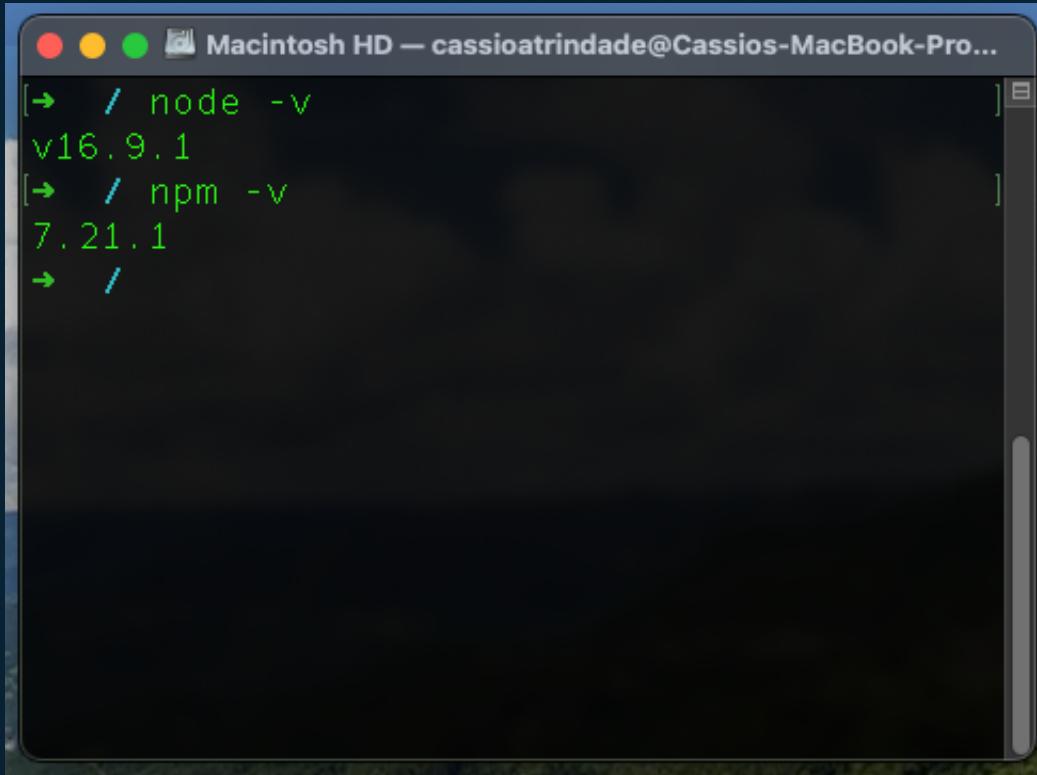
Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

For information about supported releases, see the [release schedule](#).

Instalação Node.js



A screenshot of a macOS terminal window titled "Macintosh HD — cassioatrindade@Cassios-MacBook-Pro...". The window shows the following command-line session:

```
[→ / node -v
v16.9.1
[→ / npm -v
7.21.1
→ /
```

Instalação Node.js



npm é um gerenciador de pacotes para o Node.js



Instalação Node.js

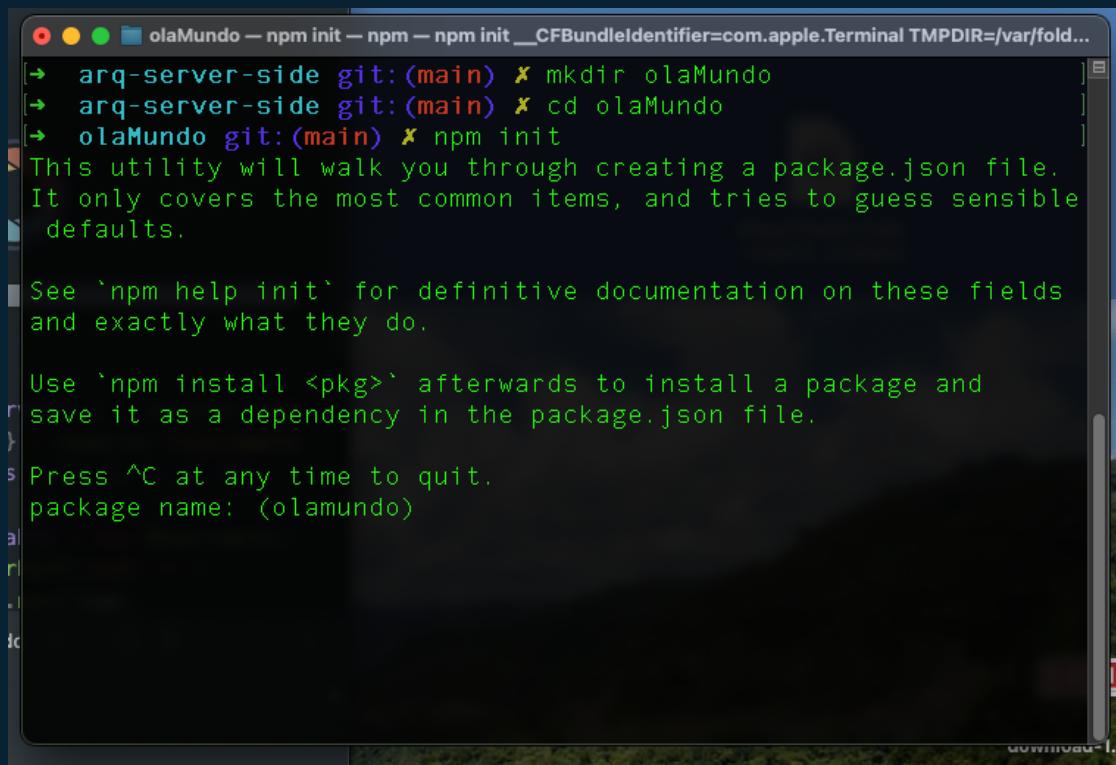


Visual Studio Code

<https://code.visualstudio.com>

IDE - Um ambiente de desenvolvimento integrado (IDE) é uma aplicação de software que ajuda os programadores a desenvolver código de software de maneira eficiente.

Olá Mundo



The screenshot shows a macOS Terminal window titled "olaMundo — npm init — npm — npm init __CFBundleIdentifier=com.apple.Terminal TMPDIR=/var/fold...". The window displays the following command history and output:

```
[→ arq-server-side git:(main) ✘ mkdir olaMundo
[→ arq-server-side git:(main) ✘ cd olaMundo
[→ olaMundo git:(main) ✘ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible
defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.
}
5 Press ^C at any time to quit.
package name: (olamundo)
a
r
.
t
```

The terminal window has a dark background with light-colored text. The title bar includes the application name and the current command being run. The main area shows the step-by-step creation of a package.json file, including prompts for the package name and a note about installing dependencies.

Olá Mundo

```
olaMundo — npm init — npm — npm init __CFBundleIdentifier=com.apple.Terminal TMPDIR=/var/fold...
package name: (olamundo)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /Users/cassioatrindade/git/arq-server-side/olaMundo/package.json:
{
  "name": "olamundo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}

Is this OK? (yes)
```

Olá Mundo

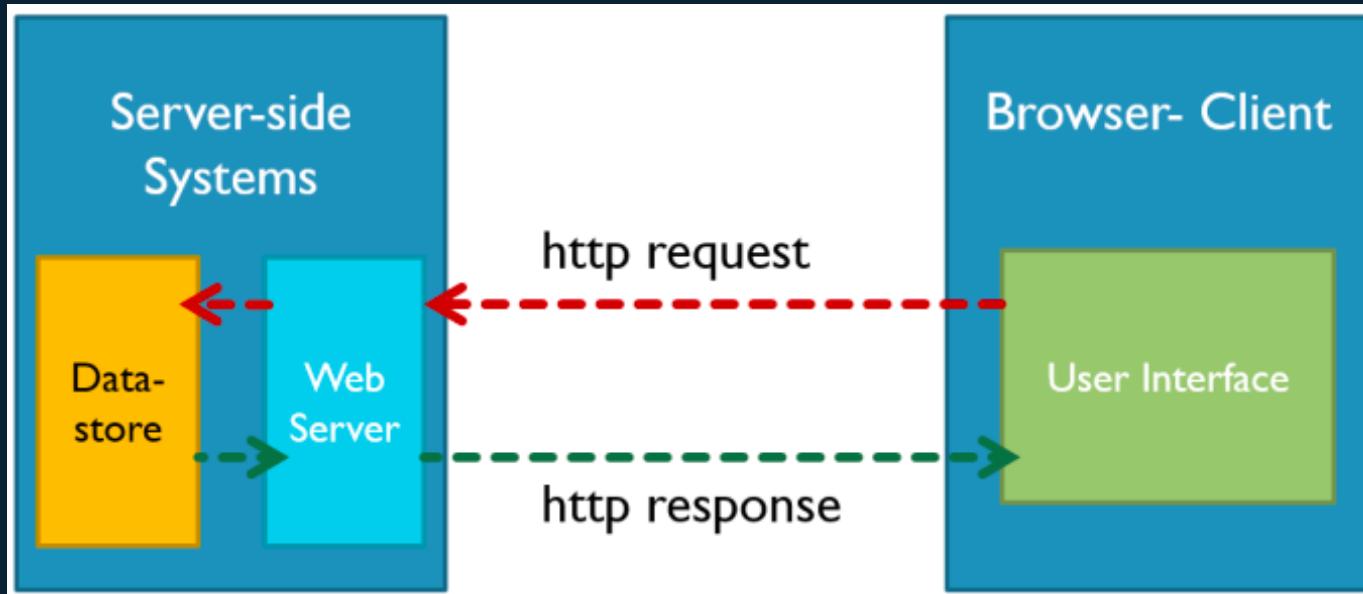
1. Crie um arquivo dentro da pasta chamado ola-mundo.js;
2. Escreva no arquivo: `console.log("Olá Mundo");`
3. Salve o arquivo;
4. Execute seu programa: `node ola-mundo.js`

```
→ olaMundo git:(main) ✘ node ola-mundo.js  
Olá Mundo
```

Programação

- Operadores aritméticos, de atribuição e de comparação:
- Estruturas de controle: if/else, for, while, for-each:
- Funções, Arrays e objetos:

Request <=> Response

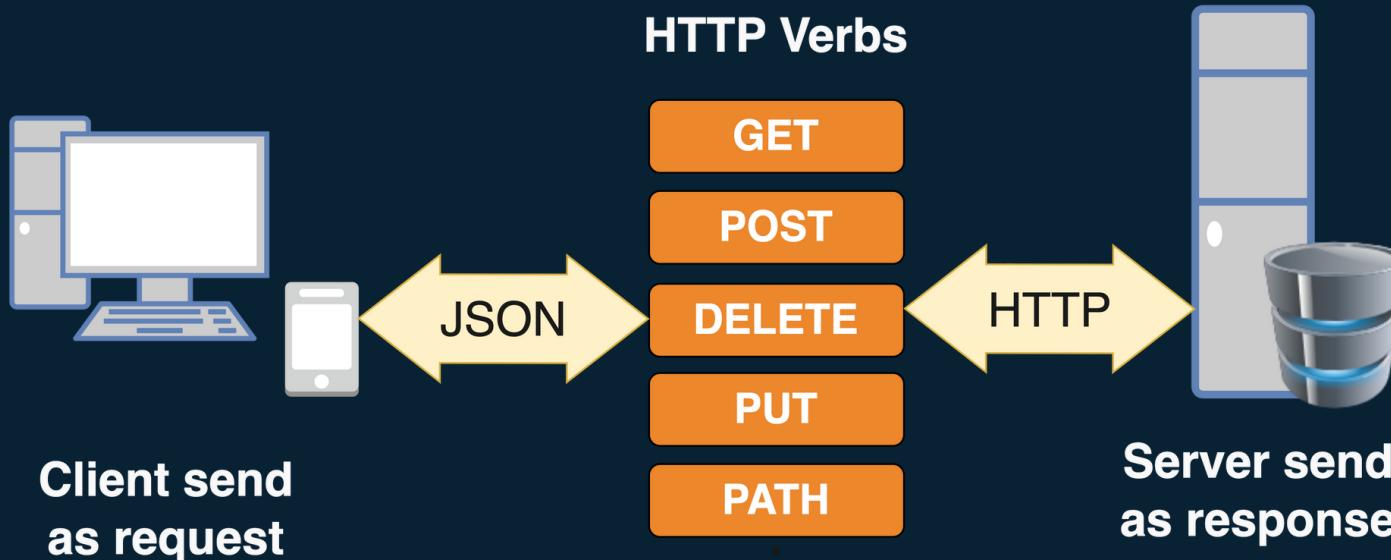


Representational State Transfer



Um estilo de arquitetura de software que define um conjunto de restrições a serem usadas para a criação de web services

API



Server com Http

```
1 const http = require('http');
2
3 const hostname = '127.0.0.1';
4 const port = 3001;
5
6 const server = http.createServer((req, res) => {
7     res.statusCode = 200;
8     res.setHeader('Content-Type', 'text/plain');
9     res.end('----- Ola Mundo -----');
10 });
11
12 server.listen(port, hostname, () => {
13     console.log(`Server running at http://${hostname}:${port}/`);
14 })
```

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.

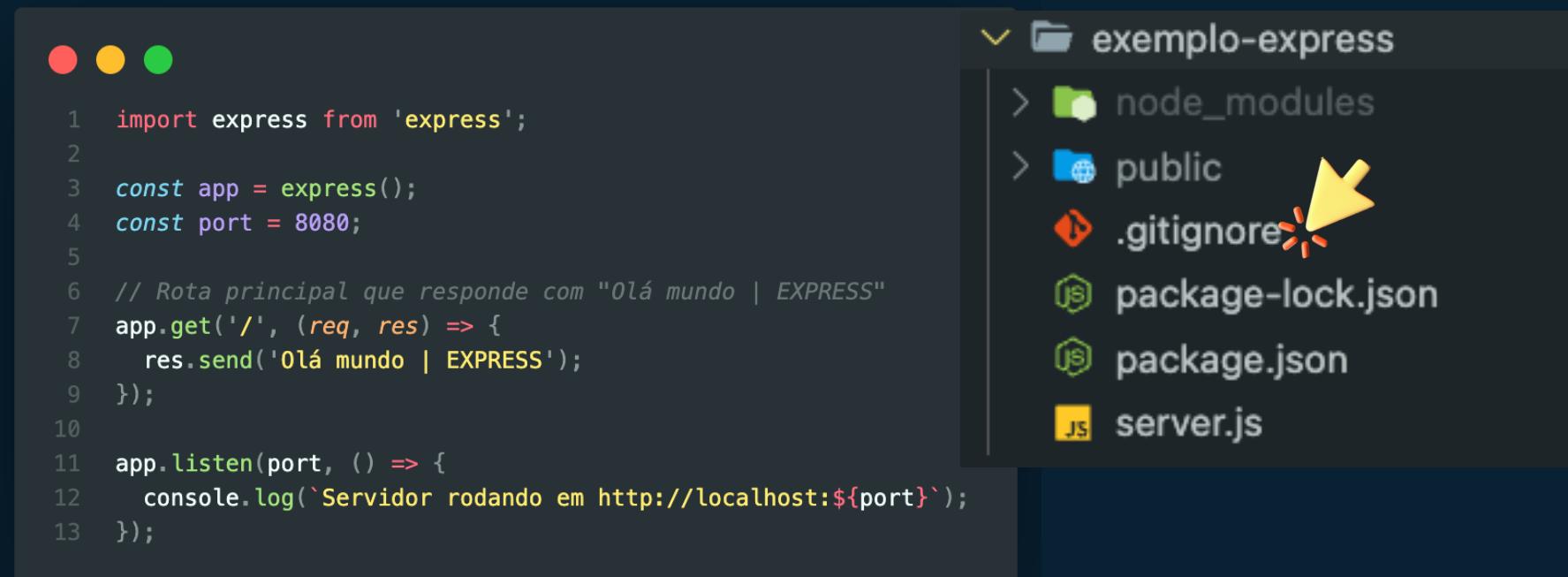
```
$ npm install express --save
```

Express application generator

```
$ npm install -g express-generator
```

Server com Express

```
$ mkdir pasta && cd pasta  
$ npm init && npm install express --save  
$ crie um arquivo server.js
```

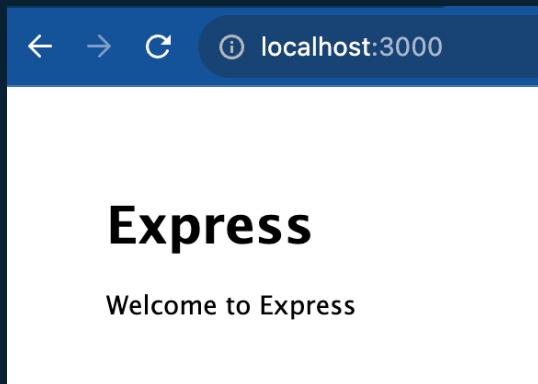


```
1 import express from 'express';  
2  
3 const app = express();  
4 const port = 8080;  
5  
6 // Rota principal que responde com "Olá mundo | EXPRESS"  
7 app.get('/', (req, res) => {  
8     res.send('Olá mundo | EXPRESS');  
9 });  
10  
11 app.listen(port, () => {  
12     console.log(`Servidor rodando em http://localhost:${port}`);  
13 });
```

The image shows a code editor interface with a dark theme. On the left, there is a vertical toolbar with three colored circles (red, yellow, green). The main area displays a snippet of JavaScript code for an Express.js application. On the right, a file explorer sidebar shows the project structure of a folder named 'exemplo-express'. The files listed are 'node_modules', 'public', '.gitignore', 'package-lock.json', 'package.json', and 'server.js'. A large yellow arrow points to the '.gitignore' file.

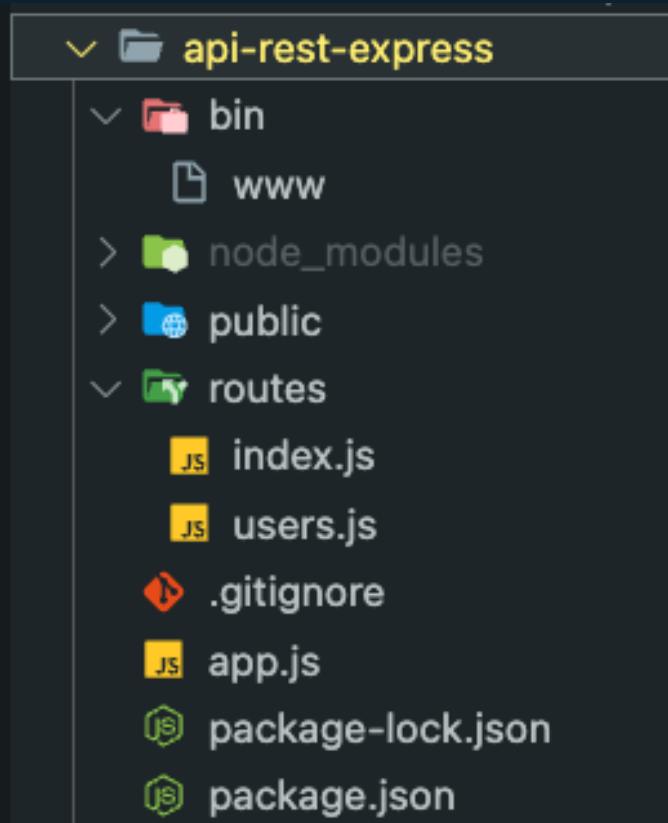
API Rest com Express

```
$ express --no-view api-rest-express  
$ cd api-rest-express  
$ npm install  
$ npm start
```



```
O ➔ api-rest-express git:(main) ✘ npm start  
> api-rest-express@0.0.0 start  
> node ./bin/www  
  
GET / 200 6.333 ms - 176  
GET /stylesheets/style.css 200 1.333 ms - 111  
GET /favicon.ico 404 4.038 ms - 150  
[]
```

API Rest com Express



API Rest com Express

Para testar APIs



POSTMAN



```
1 var express = require('express');
2 var router = express.Router();
3
4 /* GET users lista. */
5 router.get('/', function(req, res, next) {
6   res.send('Lista Usuários');
7 });
8
9 /* POST user.*/
10 router.post('/', (req, res) => {
11   res.send('Cria um Usuários')
12 });
13
14 /* DELETE User. */
15 router.delete('/',(req,res) => {
16   res.send('Deleta um usuário')
17 });
18
19 /* PUT User altera*/
20 router.put('/', (req,res) => {
21   res.send('Altera Usuario')
22 });
23
24 module.exports = router;
```

PUCRS online  uol edtech