UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Backend, Frontend e API REST

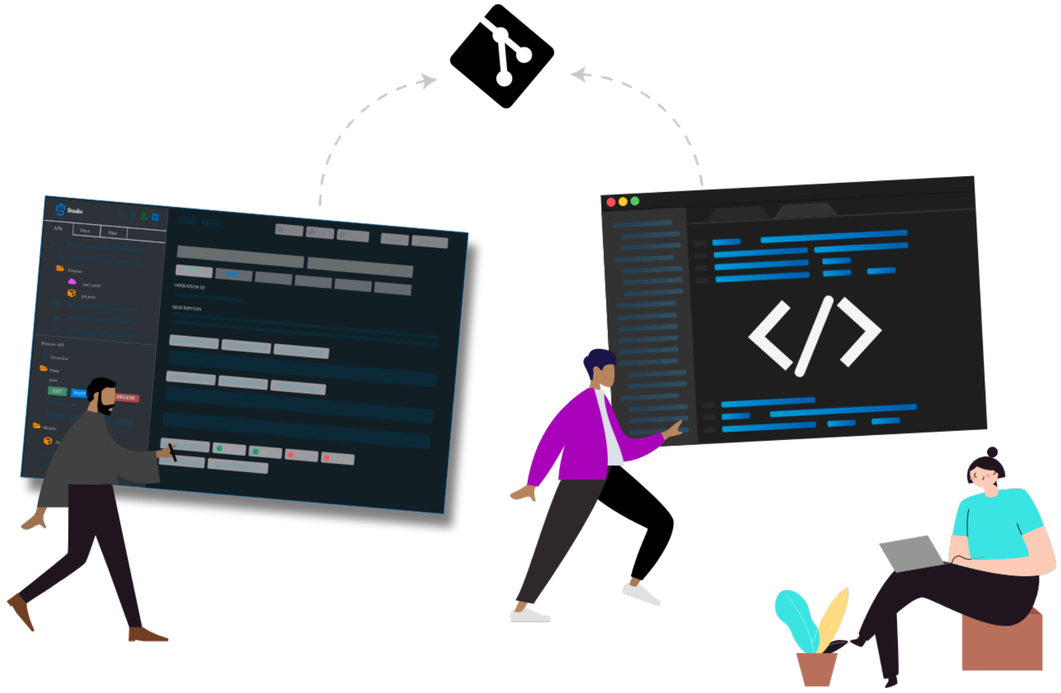
QXD0020 - Desenvolvimento de Software para Web

Prof. Bruno Góis Mateus (brunomateus@ufc.br)

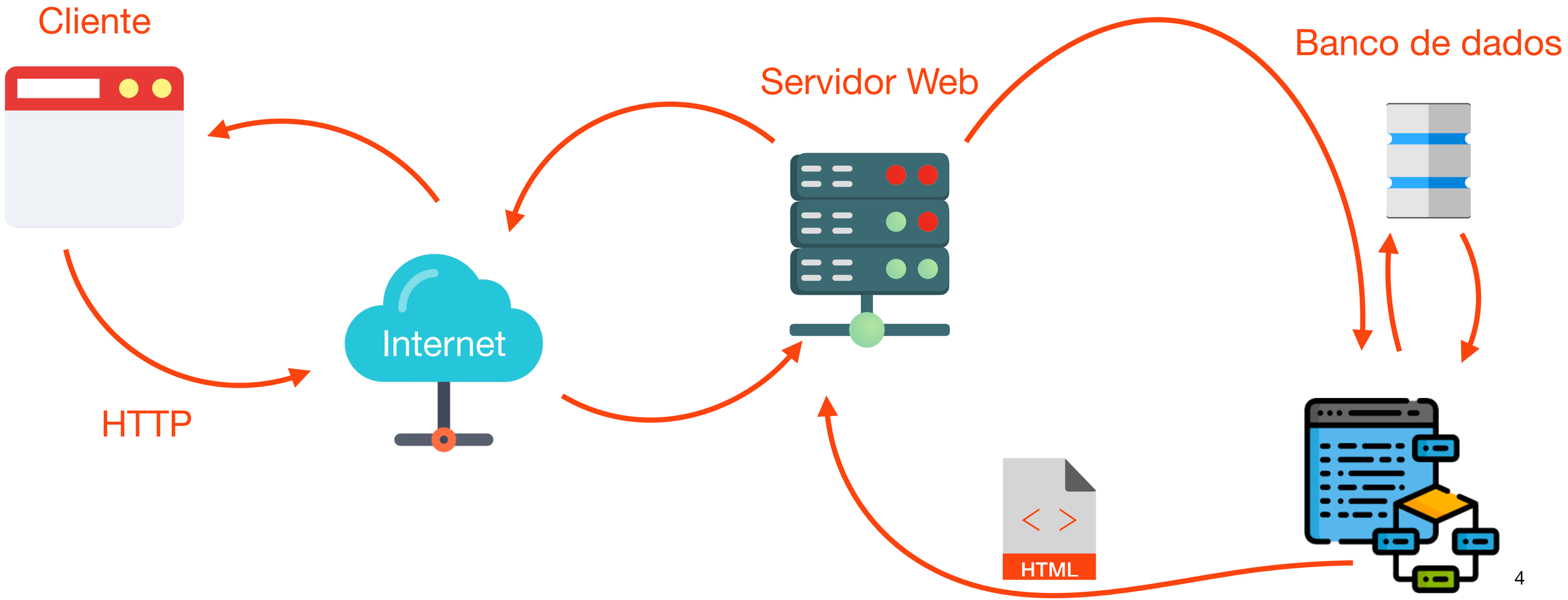
Agenda

- Introdução
- Arquitetura MVC - Model View Controller
- API REST
- Back-end vs Front-end
- Strapi

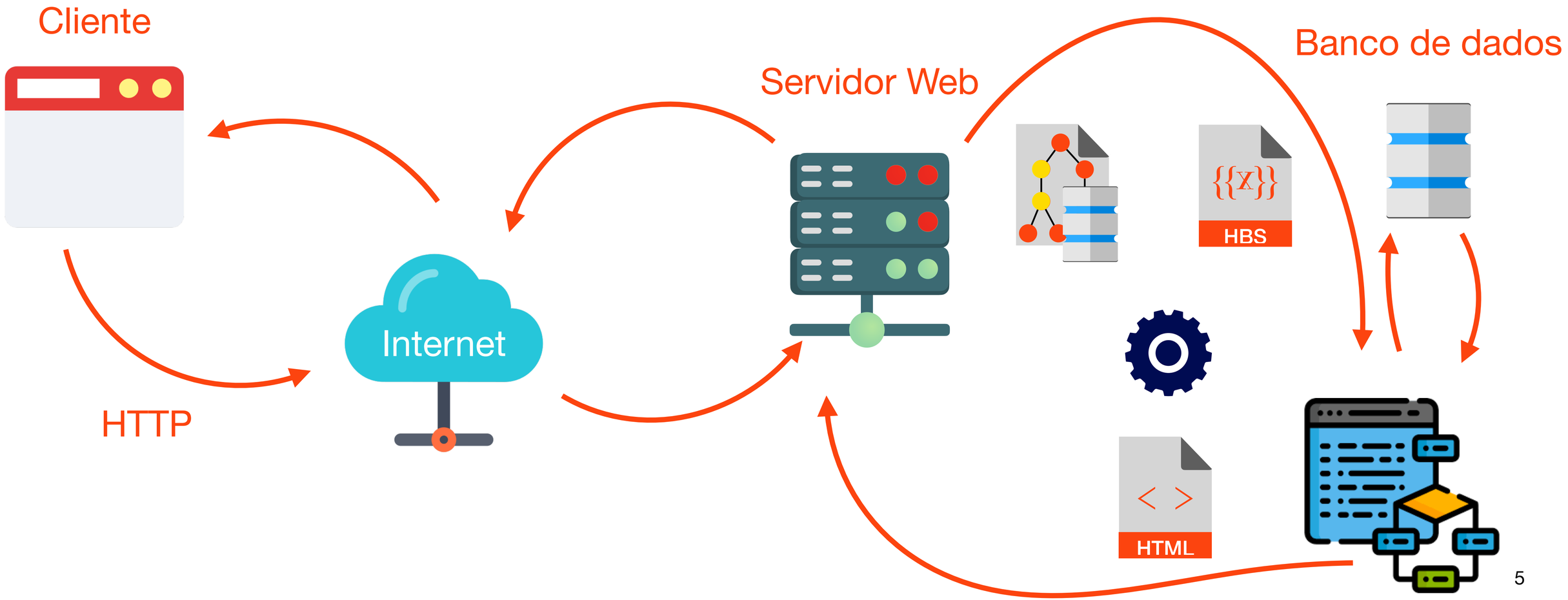
Introdução



Introdução



Introdução



Arquitetura MVC

Arquitetura MVC

Model - View - Controller

- Padrão arquitetural que se tornou popular em meados de 1970
- Separa a representação da informação da visualização da mesma
- Divide o sistema em três partes interconectadas
  - Model
  - View
  - Controller

Arquitetura MVC

Controller

Controller

- Realizam a ligação entre o usuário e o sistema
- Devem aguarda por requisições HTTP
  - Aceita entradas e converte para comandos para view ou model
  - Delega as regras de negócio para modelos e serviços
- Retorna com uma resposta significativa

Arquitetura MVC

View

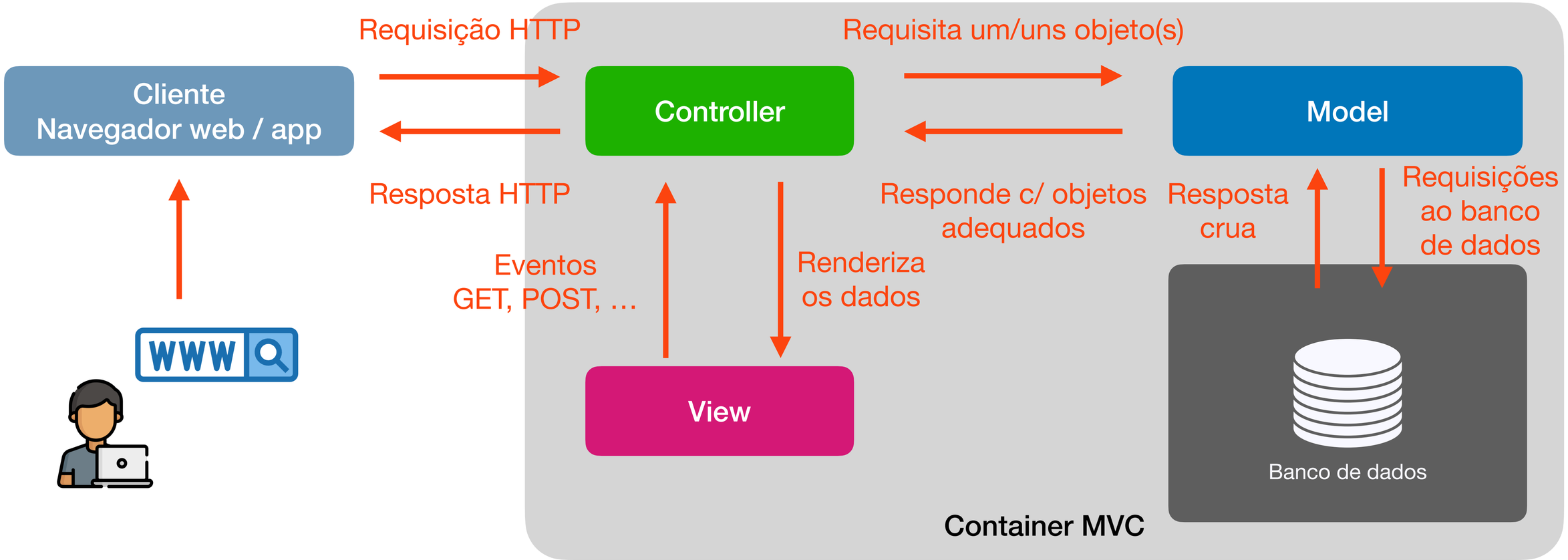
- Representação visual da nossa aplicação (GUI - Graphical User Interface)
- Mostram os dados ao usuário em forma fácil de entender baseado nas suas ações
  - Camada de interação com o usuário
  - Um mesmo conjunto de dados pode ser visualizado de n maneiras diferentes
    - Ex: Gráfico de barras , Diagrama de pizza
- Deve refletir mudanças ocorridas nos modelos

Arquitetura MVC

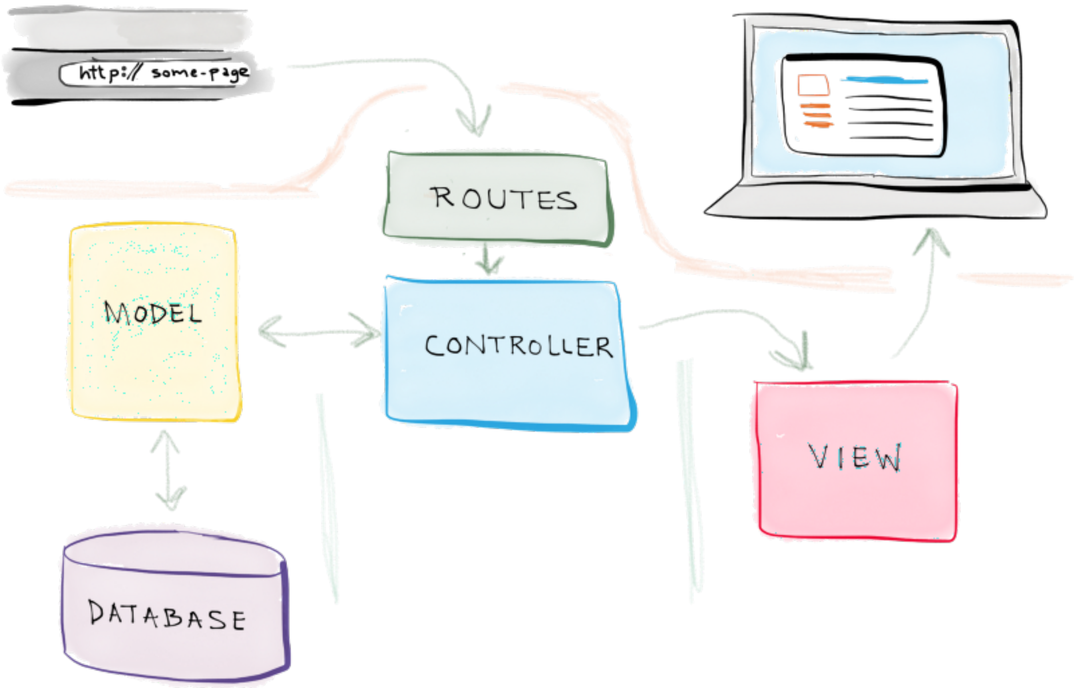
Model

- Modelo representam o conhecimento do domínio da aplicação
- Gerencia os dados, a lógica e as regras da aplicação
- Independente da interface com o usuário
- Encapsulam os dados do banco de dados
  - Tabelas

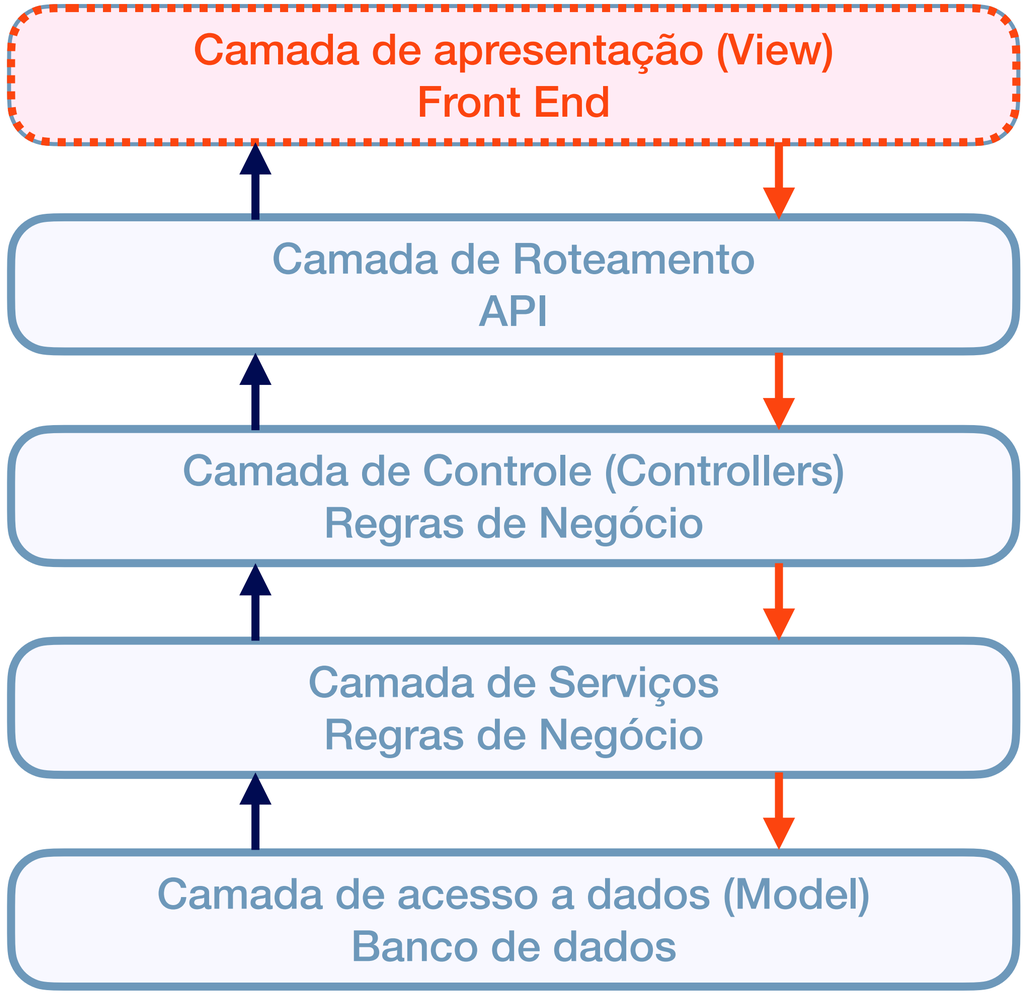
Arquitetura MVC



Arquitetura MVC



Créditos: Real Python

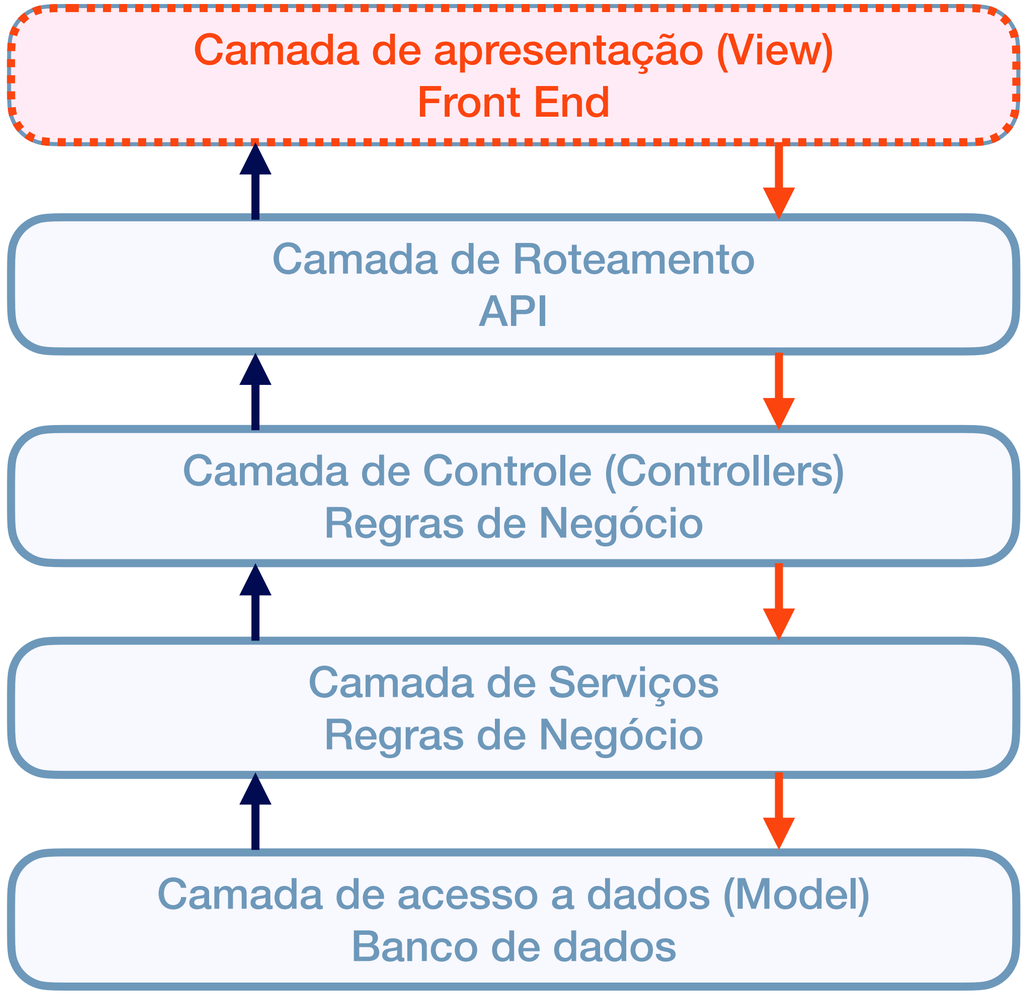


Arquitetura MVC

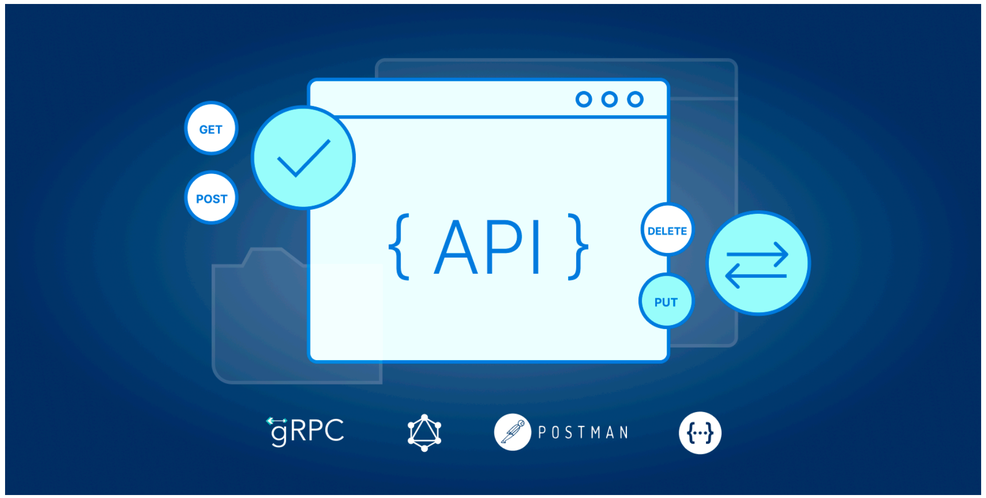
```ts
export const router = express.Router();

router.get('/', async (req, res) => {…
})
router.post('/', expressjwt({ secret: SECRET_KEY, algorithms: ["HS256"] }), isAdmin, async (req, res) => {…
});
router.get('/:id', async (req, res) => {…
});
router.put('/:id', expressjwt({ secret: SECRET_KEY, algorithms: ["HS256"] }), isAdmin, async (req, res) => {…
});
router.delete('/:id', expressjwt({ secret: SECRET_KEY, algorithms: ["HS256"] }), isAdmin, async (req, res) => {…
});
```

```ts
export class MangaService {
    private repository: Repository<Manga>
    async getAll(): Promise<Manga[]> {…
    }
    async get(id: number): Promise<Manga | null> {…
    }
    async create(title: string, number: number, price: number, coverPath?: string): Promise<Manga> {…
    }
    async delete(id: number): Promise<Manga | null> {…
    }
    async update(id: number, title: string, number: number, price: number, coverPath?: string): Promise<Manga | null> {…
    }
}
```

```ts
@Entity()
export class Manga {
    @PrimaryGeneratedColumn()
    id!: number
    @Column()
    title!: string
    @Column()
    number!: number
    @Column({nullable: true, default: ""})
    summary: string = ""
    @Column()
    price!: number
    @OneToOne(() => Cover, { nullable: true, cascade: true, onDelete: "CASCADE" })
    @JoinColumn()
    cover!: Cover
    @OneToMany(() => Comment, (comment) => comment.manga)
    @JoinColumn()
    comments?: Comment[]
}
```



API REST



API REST

Introdução

API - Application Programming Interface

- Apresenta funções e regras que permitem a interação e comunicação entre diferentes aplicativos
- Facilitam a integração de aplicativos
  - Permitindo que os desenvolvedores criem produtos digitais poderosos
- Faz a mediação entre aplicativos por meio de solicitações e respostas
  - Vários protocolos e arquiteturas podem ser utilizados
    - XML-RPC, JSON-RPC, SOAP, REST

API REST

Introdução

REST - REpresentational State Transfer

- Estilo arquitetural para sistemas distribuídos de hipermídia
- A manipulação dos recursos disponibilizados pela API é realizada através do protocolo HTTP
  - O formato JSON é o mais para representar os estados
- Conjunto de regras e boas práticas para o desenvolvimento dessas APIs
  - 6 princípios

API REST

Princípios REST

Interface Uniforme (*Uniform Interface*)

- Os recursos devem ser unicamente identificados por meio de uma URL
- Os recursos não devem ser grandes, mas devem conter toda a informação que o cliente possa necessitar

API REST

Princípios REST

Cliente-Servidor

- Trata da separação de responsabilidades
- Cliente e Servidor podem evoluir de forma independente
- O contrato entre eles se mantém intacto

API REST

Princípios REST

Sem estados (*Stateless*)

- Toda requisição realizada deve conter toda a informação necessária para que ela seja entendida
  - O servidor não deve possuir conhecimento sobre requisições feitas previamente
  - O servidor não de armazenar informações sobre as requisições

API REST

Princípios REST

Cache

- Sempre que possível o conteúdo deve ser *cacheable*
  - Seja no cliente ou no servidor
  - As respostas do devem possuir uma label indicando essa condição
    - Diretivas no cabeçalho HTTP
- O objetivo é melhorar a performance no cliente e aumentar a escalabilidade do servidor

API REST

Princípios REST

Camadas

- Arquitetura em camadas
- Devem ser projetadas de forma que nem o cliente nem o servidor saibam se eles estão se comunicando com a aplicação fim
  - Ex: Múltiplas camadas de servidores

API REST

Princípios REST

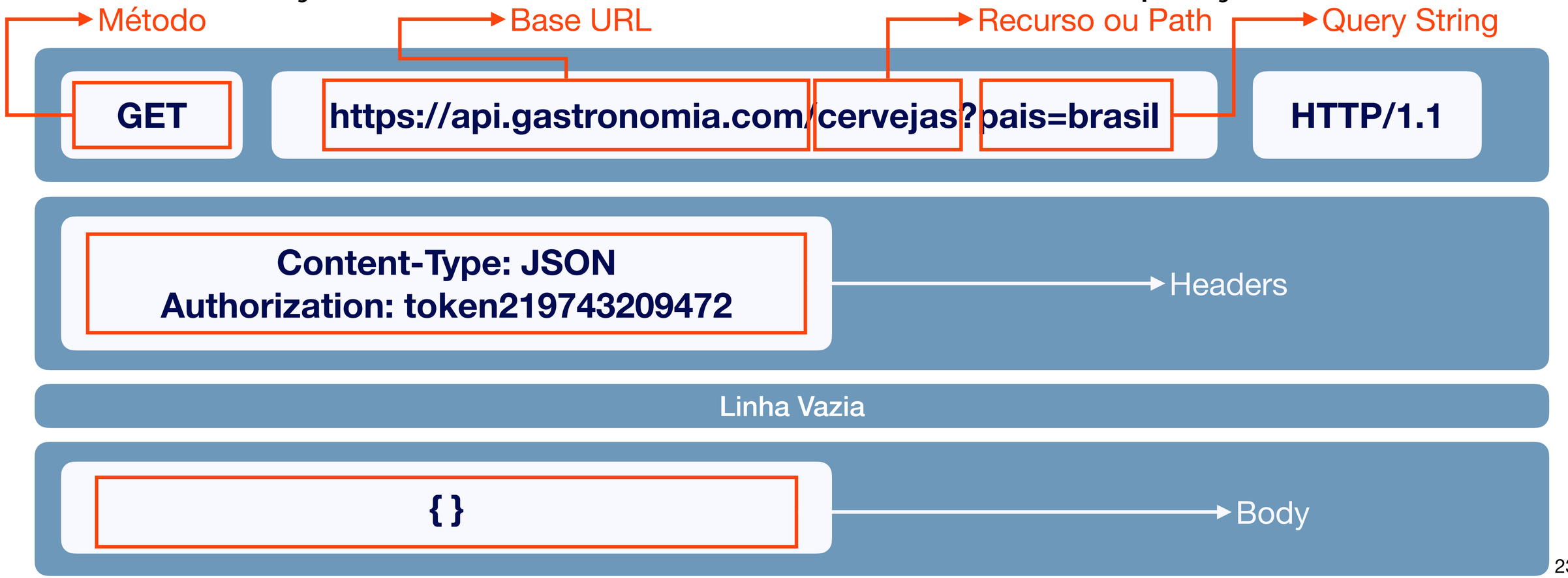
Código sobre demanda (Opcional)

- Na maioria das vezes o servidor responde com recursos estáticos, no entanto, em certos casos o servidor deve poder incluir código executável

API REST

Anatomia de uma API REST

- A comunicação com uma API REST é feita através de requisições HTTP



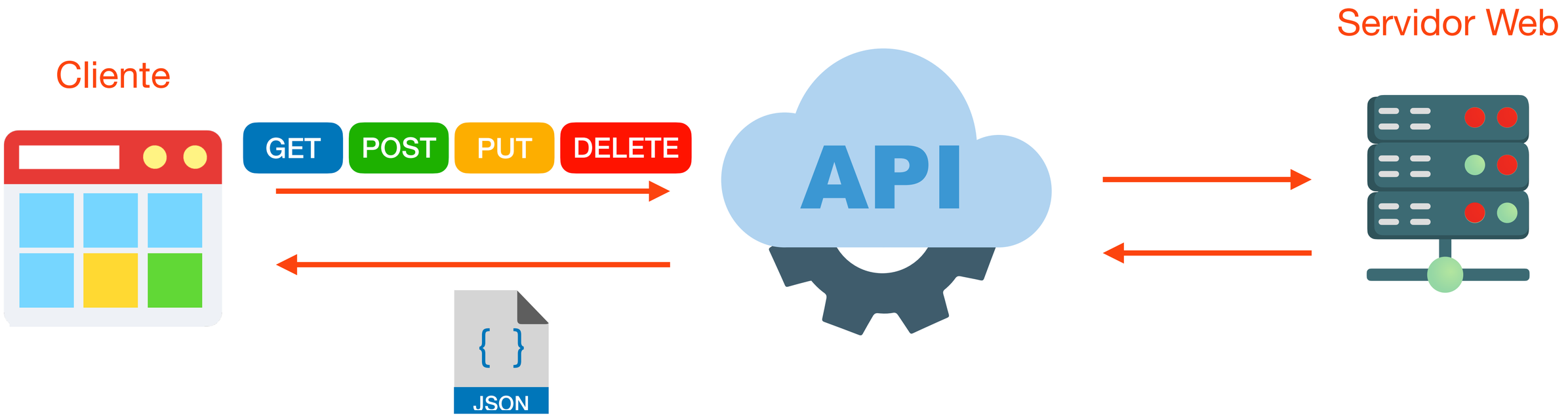
API REST

Método HTTP

- Em geral, uma API REST contempla as operações de CRUD
- Nesse ponto, o entendimentos do métodos HTTP é crucial para a construção de uma boa API

Finalidade	HTTP Method
Buscar dados em uma API	GET
Criar recurso em uma API	POST
Atualizar um recurso em uma API	PUT PATCH
Excluir um recurso em uma API	DELETE

API REST



API REST

Códigos de status HTTP

Grupo	Código	Quando
1xx - Respostas informativas	Raramente são utilizadas	Raramente são utilizadas
2xx - Envias em caso de sucesso	200 OK	Código mais utilizado. Requisição processada com sucesso
	201 Created	Indica que um novo registro foi criado. Usando em respostas a requisições POST

API REST

Códigos de status HTTP

Grupo	Código	Quando
3xx - Define respostas de redirecionamento	301 Moved Permanently	Informa que o recurso A agora é o recurso B
	304 Not modified	Resposta utilizada em cenários de cache. Informa ao cliente que a resposta não foi modificada. Portanto, o cliente pode usar a mesma versão em cache da resposta
	307 Temporary redirect	Informa sobre o redirecionamento de uma URL para outra, porém com caráter temporário

API REST

Códigos de status HTTP

Grupo	Código	Quando
4xx - Informa erros no lado do cliente	400 Bad Request	Indica que o servidor não consegue entender a requisição, devido a sintaxe ou estrutura inválida
	401 Unauthorized	Informa que existe uma camada de segurança para recurso solicitado, e que você não está utilizando as credenciais corretas nessa requisição
	403 Forbidden	Credenciais reconhecidas, porém o servidor informa que o usuário não tem os direitos de acessos necessários
	404 Not Found	Informa que o servidor não encontrou o recurso solicitado
	429 Too Many Requests	Não é tão comum, mas pode ser utilizar para informar que o cliente excedeu o limite permitido de requisições

API REST

Códigos de status HTTP

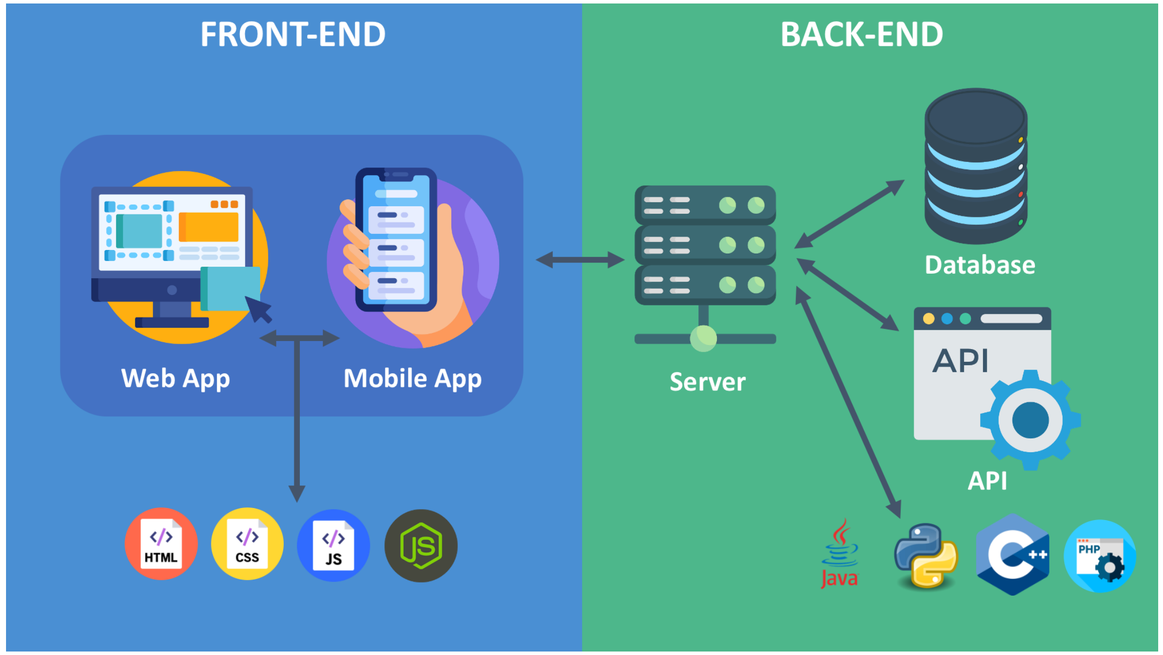
Grupo	Código	Quando
5xx - Enviadas quando ocorre um erro no lado do servidor	500 Internal Server Error	Erro mais genérico do grupo. Informa que o servidor encontrou um cenário inesperado de erro com o qual não soube lidar
	503 Service Unavailable	Normalmente é utilizado para informar que o servidor está fora do ar, em manutenção ou sobrecarregado

API REST

Exemplos de rotas

Tarefa/Funcionalidade	HTTP Method	URL
Listar mangás	GET	/mangas
Adicionar um mangá	POST	/mangas
Ver detalhes de um mangá	GET	/mangas/:id
Atualizar um mangá	PUT	/mangas/:id
Remover um mangá	DELETE	/mangas/:id

Backend vs Frontend



Back-end vs Front-end

Desenvolvedores de Frontend trabalham naquilo que o usuário consegue ver enquanto desenvolvedores de backend trabalham na infraestrutra de suporte

Backend vs Frontend

Frontend

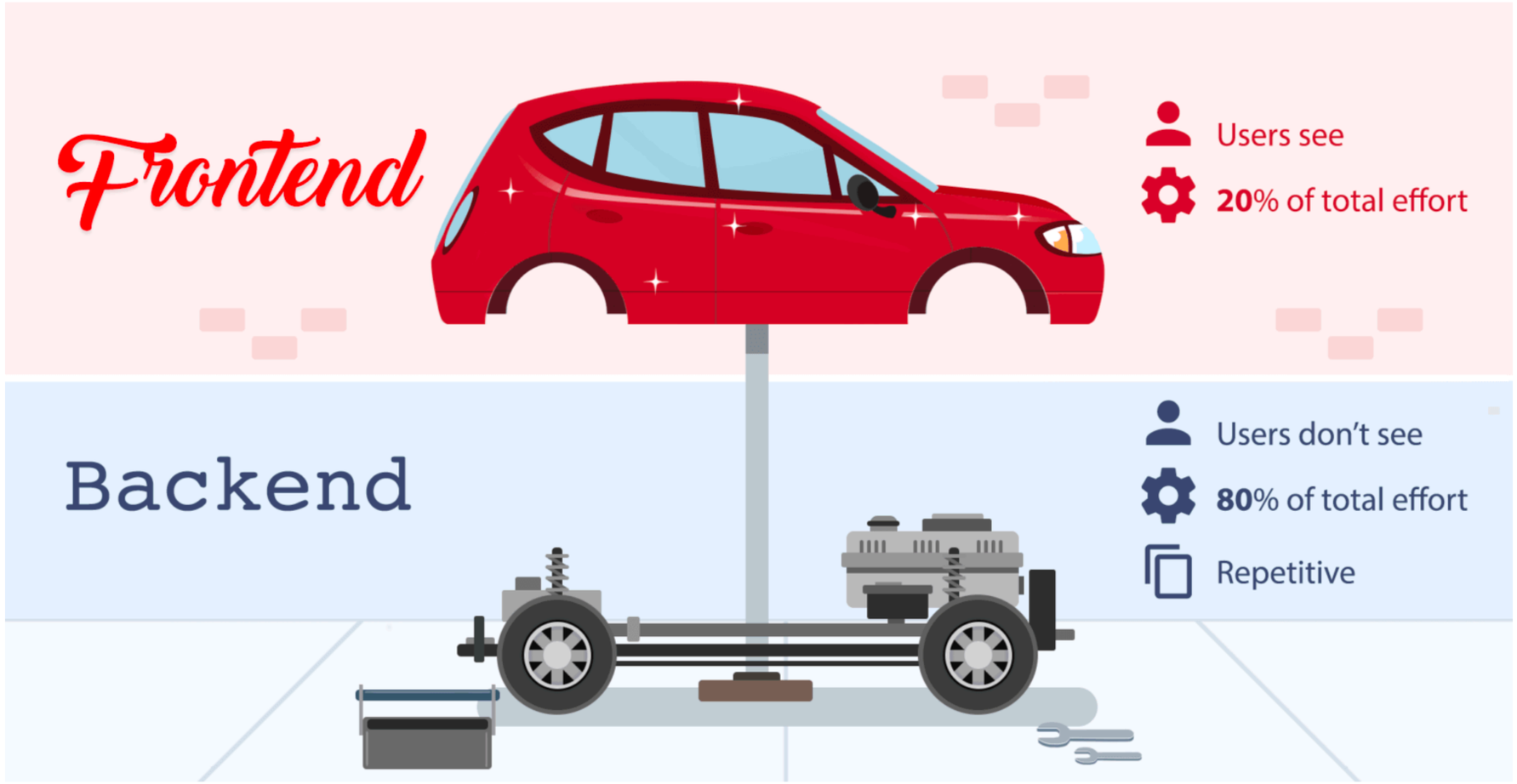
- Trabalham com a interface de usuário
- Lado do “cliente”
- Trabalham sempre pensando no usuário
- Desenvolvem elementos e funcionalidades que serão vistas pelo usuário
- Tecnologias mais utilizadas: **HTML, CSS, JS**.
  - Atualmente: React, Vue, Angular

Backend vs Frontend

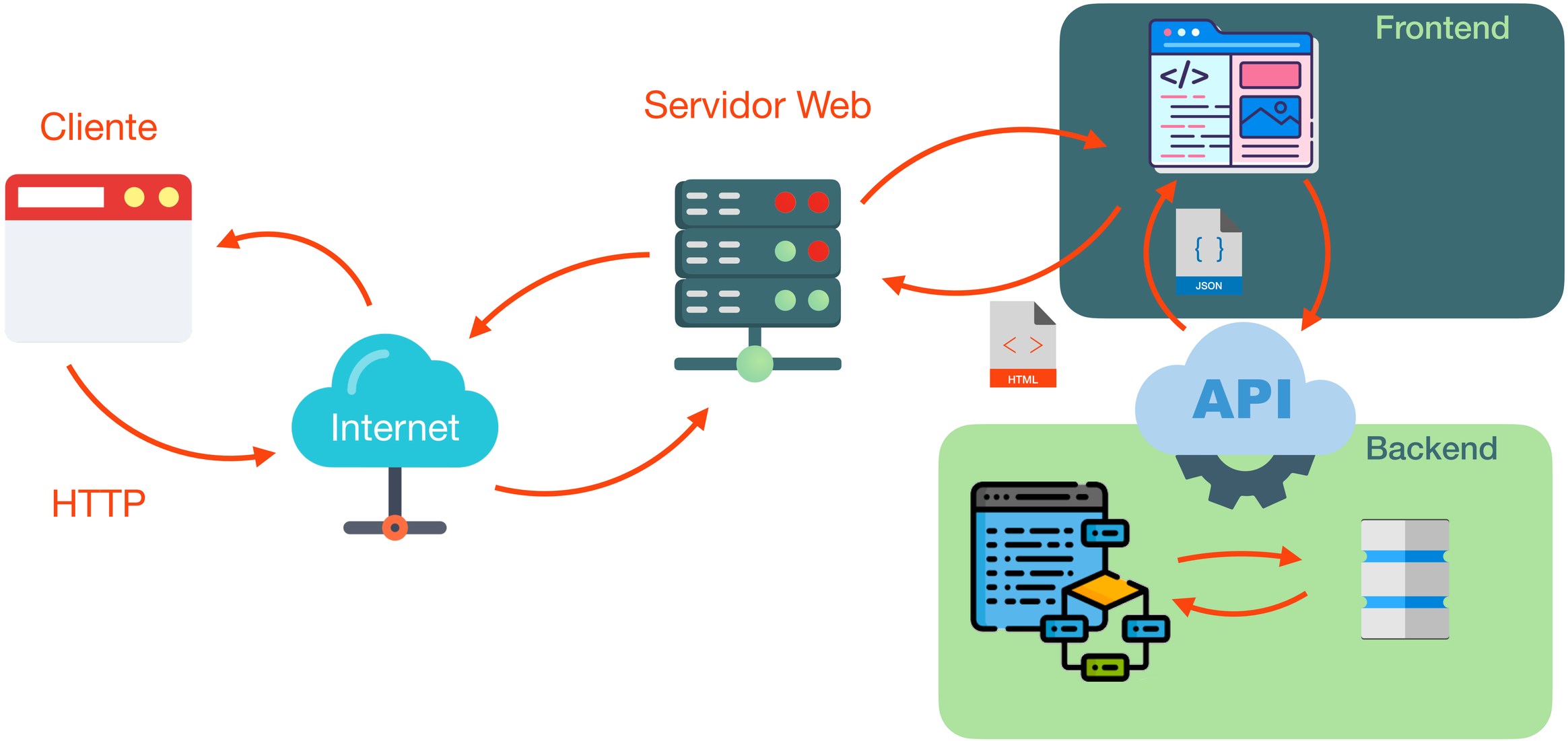
Backend

- Trabalham na parte que os usuários não conseguem ver
  - É acessado indiretamente via front-end
- Lado do servidor
- Responsável por armazenar os dados e garantir a segurança da aplicação
- Tecnologias que pode ser utilizadas:
  - JavaScript, Java, Python, Ruby e etc …

Backend vs Frontend



Backend vs Frontend



Strapi

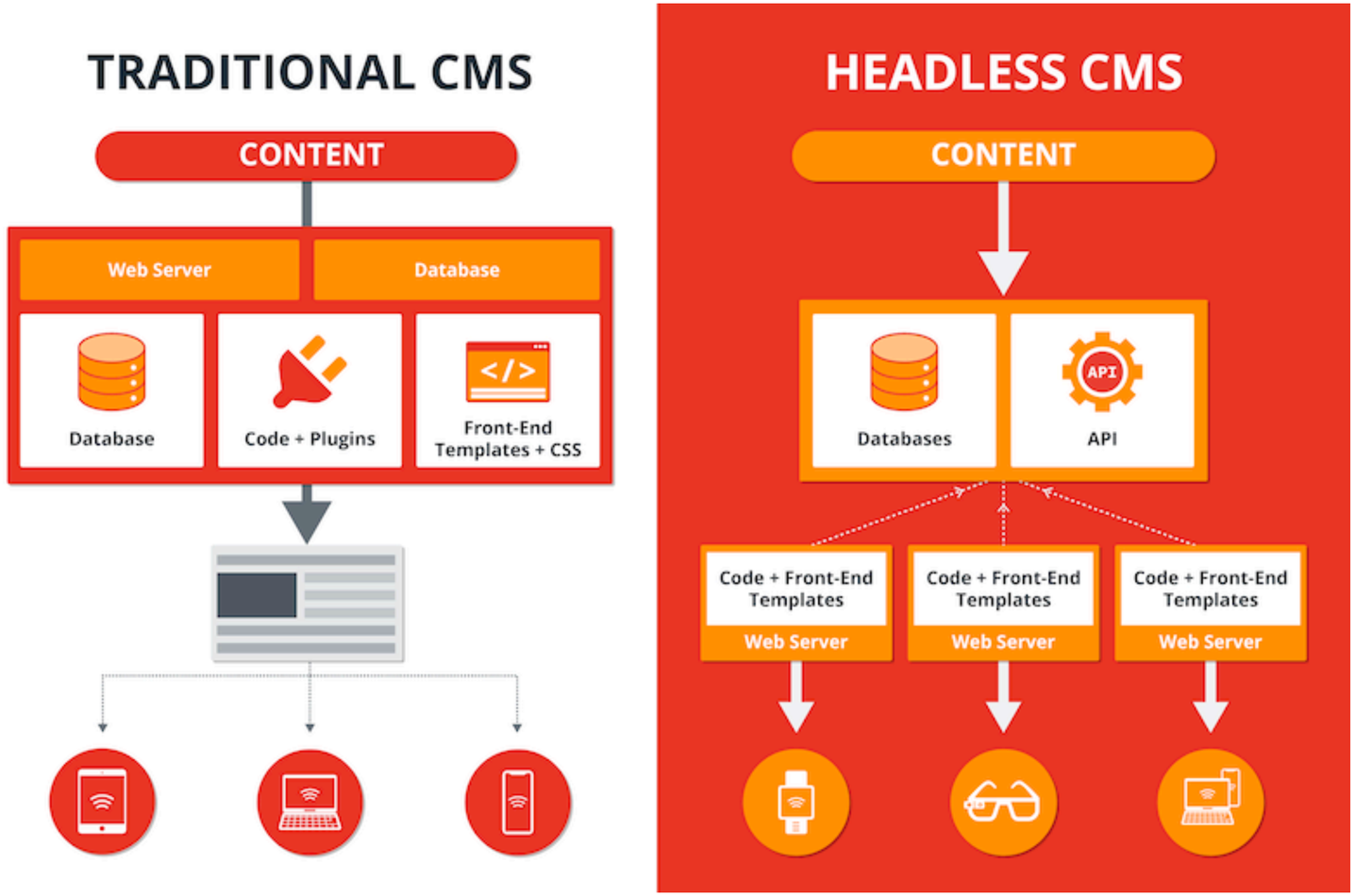
Strapi

Introdução

- Open-source Headless *Content Management System (CMS)*
  - Sistema de gerenciamento de conteúdo “Sem cabeça”
- Neste contexto, um CMS é divido em duas partes
  - head: onde o conteúdo é apresentado
  - body: onde o conteúdo é criado e armazenado

Strapi

Traditional vs Headless CMS



Fonte: https://willianjusten.com.br/introducao-ao-strapi-headless-cms

Strapi

Vantagens de um Headless CMS

- Edição de conteúdo mais rápida
  - CMS tradicional: Gasta-se recursos na edição e renderização de conteúdo
  - Headless CMS: foco somente gerenciamento de conteúdo
- Gerenciamento de conteúdo para vários canais
  - Headless CMS não são atrelados a um único canal de apresentação (ex: websites).
  - Mais fácil gerenciar o conteúdo e atender diversos canais
- Developer flexibility
  - Como o conteúdo é disponibilizado via API, desenvolvedores pode escolher que tecnologia usar para consumir o conteúdo e apresentá-lo ao usuário final

Strapi

Vantagens de um Headless CMS

- Developer flexibility
  - Como o conteúdo é disponibilizado via API
  - Para consumir o conteúdo via API e apresentá-lo ao usuário final qualquer tecnologia pode ser adotada
- Melhor segurança
  - Menor área de ataque devido a separação entre conteúdo e apresentação

Strapi

Vantagens de um Headless CMS

- Fácil de escalar
  - Permite gerenciar o conteúdo usando uma única fonte de verdade (*single source of truth*)
  - Permite a troca de ferramentas de desenvolvimento a qualquer momento
  - É possível executá-la em serviços de hospedagem na nuvem de alta-perfomance como Vercel and Netlify

Strapi

CMS tradicional vs Headless CMS

Funcionalidade	CMS Tradicional	Headless CMS
Arquitetura	Monolítica	Frontend e Backend desacoplados
Desenvolvimento do Frontend	Baseado em templates	Liberdade total. Inclusive de ferramentas.
Design	Focado em websites	Qualquer tipo de aplicação
Otimizações com foco em alta perfomance	Necessárias	Mais fáceis de serem implementadas
Escalabilidade	Linear	Escalável por design
Experiência do desenvolvedor	Código “legado”	Tecnologias atuais
Seguranças	Maior superfície de ataque	Menor superfície de ataque
Iterações de desenvolvimento	Ciclos longos	Ciclos curtos

Strapi

Funcionalidades

- Suporte a múltiplos bancos de dados
  - SQLite, MongoDB, MySQL, Postgres
- GraphQL or RESTful
  - API geradas podem ser consumidas via REST ou GraphQL.
- 100% Javascript
  - Uma única linguagem para o frontend e o backend
- Web hooks
- Documentação auto-gerada

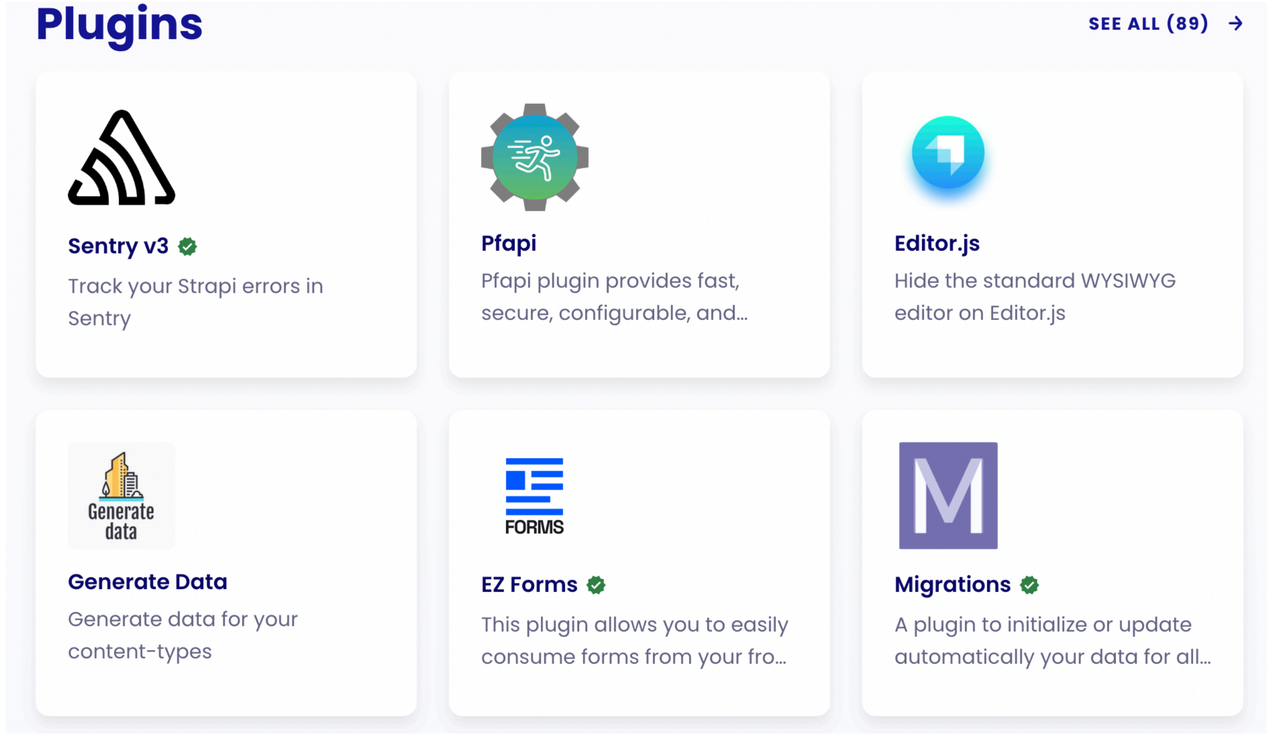
Strapi

Funcionalidades

- JWT authentication
- Custom Roles & Permissions
- Authentication & Permissions
  - Endpoint protegidos permitindo ou não o acesso de acordo com os papéis do usuário
- Customizable API
- Internationalization
- Built-in Emailing

Strapi

Market Place



Strapi

Plugins recomendados

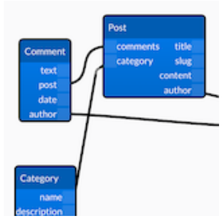

```
  data": {
   "id": 1,
   "attributes": {
     "title": "Lorem Ipsum",
     "createdAt": "2022-02-11T01:51:49.902Z",
     "updatedAt": "2022-02-11T01:51:52.797Z",
     "publishedAt": "2022-02-11T01:51:52.794Z",
     "ipsum": {
       "data": {
         "id": 2,
         "attributes": {
           "title": "Dolor sat",
           "createdAt": "2022-02-15T03:45:32.669Z",
           "updatedAt": "2022-02-17T00:30:02.573Z",
           "publishedAt": "2022-02-17T00:07:49.491Z",
         },
       },
     },
   },
  },
  "meta": {},
}
```

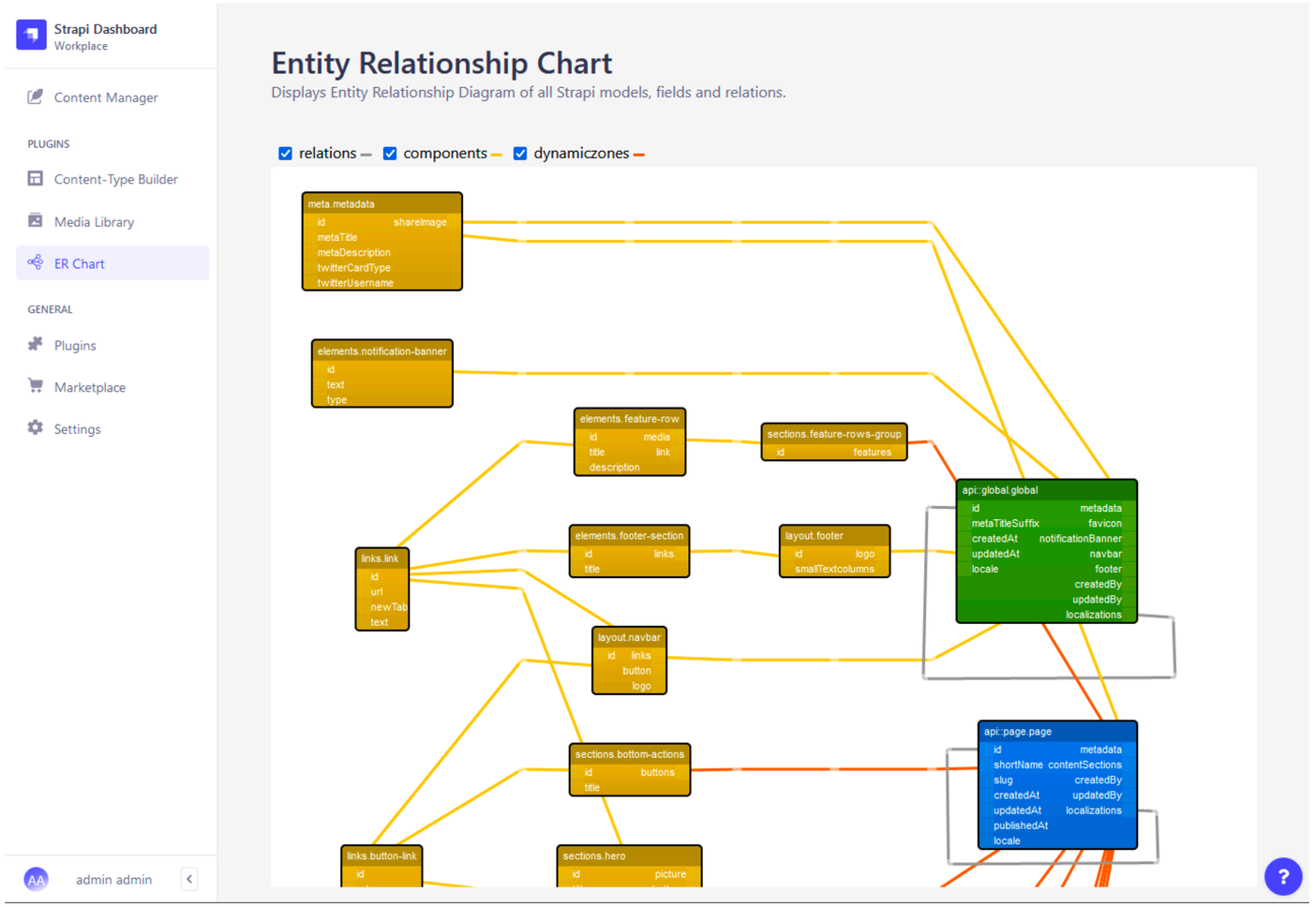


```
{
  "data": {
    "id": 1,
    "title": "Lorem Ipsum",
    "createdAt": "2022-02-11T01:51:49.902Z",
    "updatedAt": "2022-02-11T01:51:52.797Z",
    "publishedAt": "2022-02-11T01:51:52.794Z",
    "ipsum": {
      "id": 2,
      "title": "Dolor sat",
      "createdAt": "2022-02-15T03:45:32.669Z",
      "updatedAt": "2022-02-17T00:30:02.573Z",
      "publishedAt": "2022-02-17T00:07:49.491Z",
   },
  },
  "meta": {},
}
```

Strapi

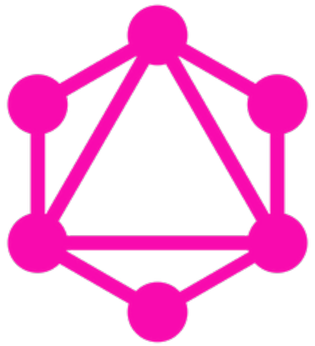
Plugins recomendados





Strapi

Plugins não testados



GraphQL

FCM

Config Sync

Email Designer

REST Cache

Stripe Payments

Documentation

Referências

- What Is the Difference Between Front-End and Back-End Development?
- Front End vs. Back End: What’s the Difference?
- Frontend vs. backend: what's the difference?
- O que é a API REST e como ela difere de outros tipos?
- A anatomia de uma API RESTful
- What is REST
- API REST: o que é e como montar uma API sem complicação?

Referências

- Introdução ao Strapi - Headless CMS
- What is a Headless CMS?
- O que é um headless CMS?
- Strapi Official Website
- Strapi Marketplace

Por hoje é só