ENTENDENDO

1 - SOLID

2 - ARQUITETURA HEXAGONAL

3 - CLEAN ARCHITECTURE

4 - ARQUITETURA BASEADA EM BFF

5 - ARQUITETURA MVVM

6 - ARQUITETURA MICRO-FRONTENDS

ENTENDENDO TUDO
EM 10 MINUTOS

Desvendando

SOLID Principles

Isaac Gomes

TS

o que é SOLID ?

S O L I D

SOLID é um acrônimo que consolida 5 itens que são considerados como boas práticas no mundo do desenvolvimento orientado a objetos.

Isaac Gomes

o que é cada acrônimo ?

S => Single Responsibility Principle

O => Open/Closed Principle

L => Liskov Substitution Principle

I => Interface Segregation Principle

D => Dependency Inversion Principle

Isaac Gomes

SOLID

Single Responsibility Principle

Cada classe deve ter uma única responsabilidade ou razão para mudar. Isso significa que uma classe deve ter apenas uma responsabilidade ou uma função específica para manter a coesão.

Isaac Gomes

SOLID

Single Responsibility Principle

Cada classe deve ter uma única responsabilidade ou razão para mudar. Isso significa que uma classe deve ter apenas uma responsabilidade ou uma função específica para manter a coesão.

Isaac Gomes

SOLID

Single Responsibility Principle

```ts
class UserService {
  createUser(name: string, email: string): void {
    this.saveUserToDatabase(name, email);
  }

  private saveUserToDatabase(name: string, email: string){}

  sendWelcomeEmail(email: string){}
}
```

note quantas responsabilidades temos em um unico local (criar, persistir e enviar email)

concorda que persistir os dados e enviar email não deviria influenciar na criação?

Isaac Gomes

TS

SOLID

Single Responsibility Principle

```ts
class UserService {
  private userRepository: UserRepository;
  private emailService: EmailService;

  constructor(
    userRepository: UserRepository,
    emailService: EmailService
  ){
    this.userRepository = userRepository;
    this.emailService = emailService;
  }

  createUser(name: string, email: string): void {
    this.userRepository.save(name, email);
    this.emailService.sendWelcomeEmail(email);
  }
}
```

note que agora quebramos essas responsabilidades em outros services e agora esse service de user tem somente um motivo para mudar

Isaac Gomes

TS

SOLID

Open/Closed Principle

Entidades de software (classes, módulos, funções, etc.) devem estar abertas para extensão, mas fechadas para modificação. Isso implica que o comportamento de uma entidade pode ser estendido sem alterar seu código-fonte original.

Isaac Gomes

SOLID
Open/Closed Principle

```ts
class DiscountCalculator {
  calculateDiscount(product: { type: string, price: number }): number {
    if (product.type === 'electronics') {
      return product.price * 0.1;
    }
    if (product.type === 'clothing') {
      return product.price * 0.2;
    }
    return 0;
  }
}
```

note que qualquer adição resulta em uma modificação na class

Isaac Gomes

TS

SOLID
Open/Closed Principle

```ts
class ClothingDiscount implements DiscountStrategy {
  calculate(price: number): number {
    return price * 0.2;
  }
}
```

```ts
class DiscountCalculator {
  private discountStrategies: { [key: string]: DiscountStrategy } = {};
  constructor() {
    this.discountStrategies['electronics'] = new ElectronicsDiscount();
    this.discountStrategies['clothing'] = new ClothingDiscount();
  }

  calculateDiscount(product: { type: string, price: number }): number {
    const strategy = this.discountStrategies[product.type];
    if (!strategy) return 0;
    return strategy.calculate(product.price);
  }
}
```

agora note criamos um strategy e basicamente se quisermos adicionar um novo basta implementar e injetar

Isaac Gomes

TS

SOLID

Liskov Substitution Principle

Objetos de uma classe derivada devem poder substituir objetos de uma classe base sem alterar a funcionalidade do programa. Isso significa que as subclasses devem ser substituíveis por suas classes base.

Isaac Gomes

SOLID
Liskov Substitution Principle

```ts
class Bird {
  fly(): void {
    console.log('Flying...');
  }
}

class Eagle extends Bird {}

class Penguin extends Bird {
  fly(): void {
    throw new Error('Penguins cannot fly!');
  }
}

function makeBirdFly(bird: Bird) {
  bird.fly();
}
```

Vamos considerar um exemplo de uma classe base Bird e duas subclasses Eagle e Penguin. Ambas as subclasses herdam de Bird, mas um pinguim não pode voar, quebrando assim o LSP.

Isaac Gomes

TS

SOLID
Liskov Substitution Principle

```ts
interface Flyable {
  fly(): void;
}

class Bird {}

class Eagle extends Bird implements Flyable {
  fly(): void {
    console.log('Flying...');
  }
}

class Penguin extends Bird {}

function makeBirdFly(bird: Flyable) {
  bird.fly();
}
```

Para resolver este problema, podemos usar a composição em vez da herança, ou podemos introduzir uma interface para classes de aves voadoras.

Isaac Gomes

TS

SOLID

Interface Segregation Principle

Uma interface deve ter apenas os métodos que são relevantes para a classe que a implementa. Classes não devem ser forçadas a implementar métodos que não utilizam.

Isaac Gomes

SOLID

Interface Segregation Principle

```ts
interface Worker {
  work(): void;
  attendMeeting(): void;
  fileReport(): void;
}

class Developer implements Worker {
  work(): void {}

  attendMeeting(): void {}

  fileReport(): void {
    // Developer não precisa implementar isso
    throw new Error('Method not implemented.');
  }
}

class Manager implements Worker {
  work(): void {}

  attendMeeting(): void {}

  fileReport(): void {
    // Manager não precisa implementar isso
    throw new Error('Method not implemented.');}
}
```

veja como temos uma **interface** muito generica acabamos com **metodos** que não dizem respeito a quele **contexto**

Isaac Gomes

TS

SOLID

Interface Segregation Principle

```ts
interface Workable {
  work(): void;
}

interface MeetingAttendee {
  attendMeeting(): void;
}

interface ReportFiler {
  fileReport(): void;
}

class Developer implements Workable, MeetingAttendee {
  work(): void {}
  attendMeeting(): void {}
}

class Manager implements MeetingAttendee, ReportFiler {
  attendMeeting(): void {}
  fileReport(): void {}
}
```

agora quebramos as **interfaces** e conseguimos fazer uma **implementação** que faz sentido no contexto

Isaac Gomes

TS

SOLID

Dependency Inversion Principle

Dependa de abstrações, não de concretizações. Isso significa que os módulos de alto nível não devem depender de módulos de baixo nível, mas ambos devem depender de abstrações. Além disso, as abstrações não devem depender de detalhes, mas os detalhes devem depender de abstrações.

Isaac Gomes

SOLID
Dependency Inversion Principle

```ts
class EmailService {
  sendEmail(to: string, subject: string, body: string): void {
    console.log(`Sending email to ${to}: ${subject} - ${body}`);
  }
}

class OrderProcessor {
  private emailService: EmailService;

  constructor() {
    this.emailService = new EmailService();
  }

  processOrder(orderId: string): void {
    this.emailService.sendEmail(
      'customer@example.com',
      'Order Confirmation',
      `Your order ${orderId} has been processed.`
    );
  }
}
```

veja com instanciamos diretamente acabamos tendo esse forte acoplamento

Isaac Gomes

TS

SOLID
Dependency Inversion Principle

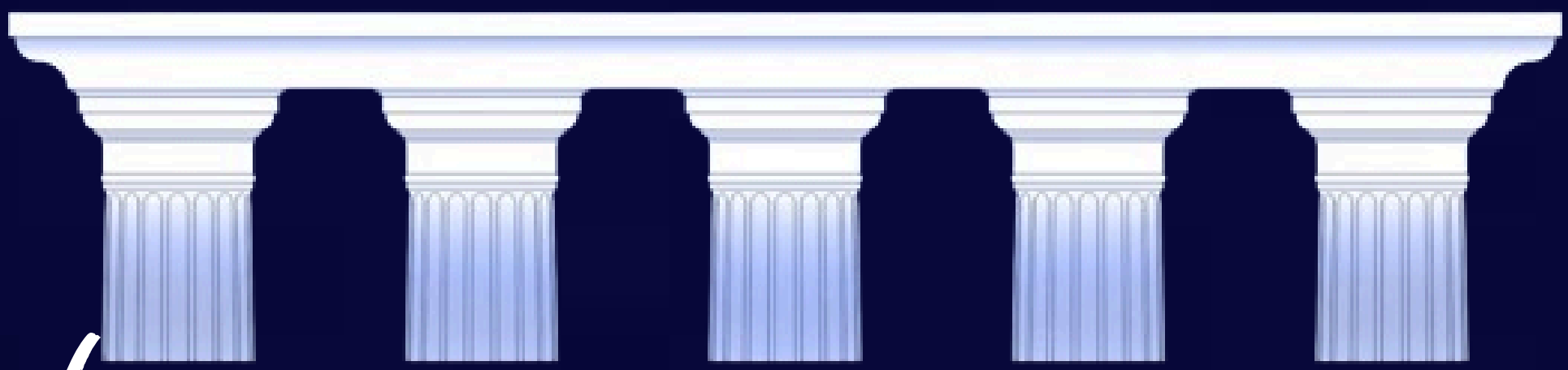
```ts
interface NotificationService {
  sendNotification(to: string, subject: string, body: string): void;
}

class EmailService implements NotificationService {
  sendNotification(to: string, subject: string, body: string): void {

  }
}

class OrderProcessor {
  constructor(
    private notificationService: NotificationService
  ) {}

  processOrder(orderId: string): void {
    this.notificationService.sendNotification(
      'customer@example.com',
      'Order Confirmation',
      `Your order ${orderId} has been processed.`
    );
  }
}
```

para **invertemos a dependências** criamos uma **interface** agora dependemos da interface e podemos invejar qualquer **implementação** que siga o que a interface requer

Isaac Gomes

TS

S O L I D



agora as letras começam
a tornar princípios
simples de entender

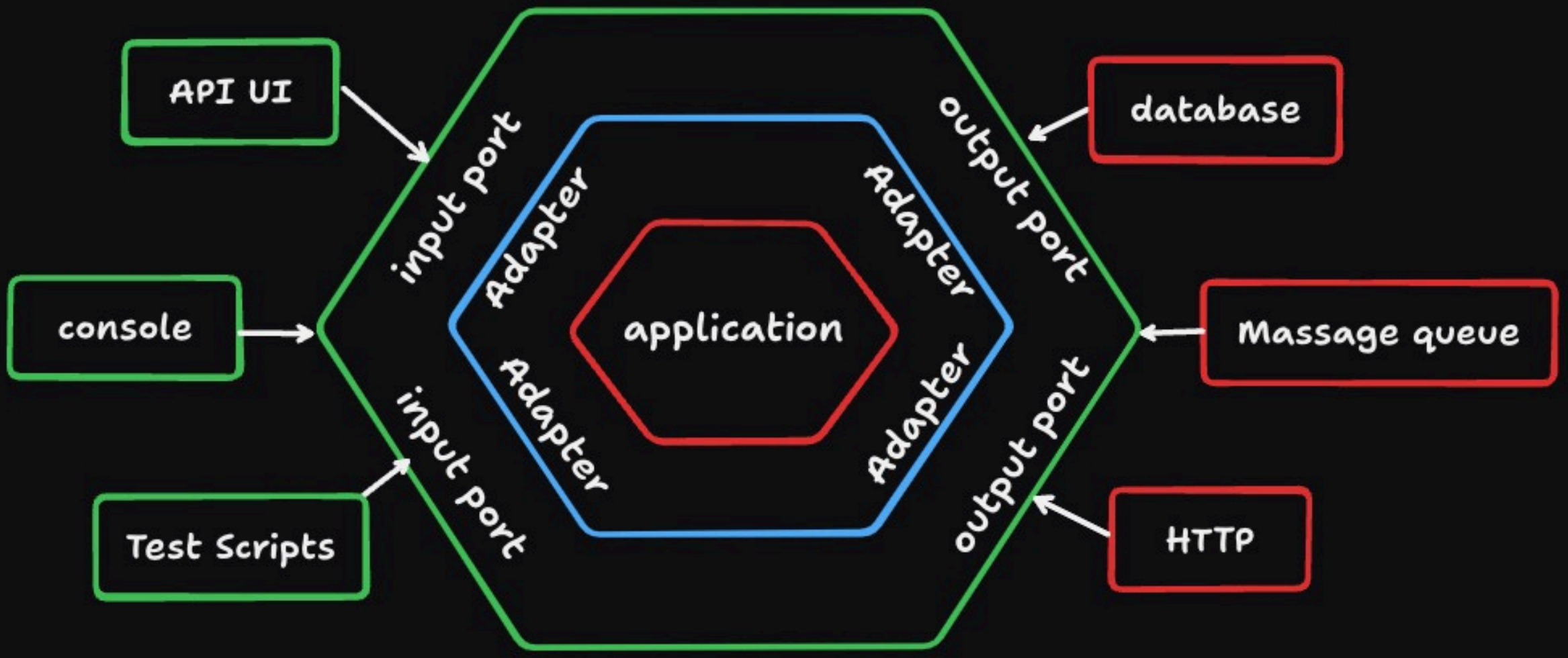
Isaac Gomes

TS

Desvendando arquitetura hexagonal

Ports and Adapters

ports and adapters



exemplo em Typescript

Isaac Gomes

o que é arquitetura de software?

quando pensamos em **arquitetura de software** é comum pensarmos em pastas ou até mesmo em nomes como **MVC, MVVM, CLEAN ARCH, PORTS AND ADAPTERS...**

mas, o que de fato é arquitetura de software?

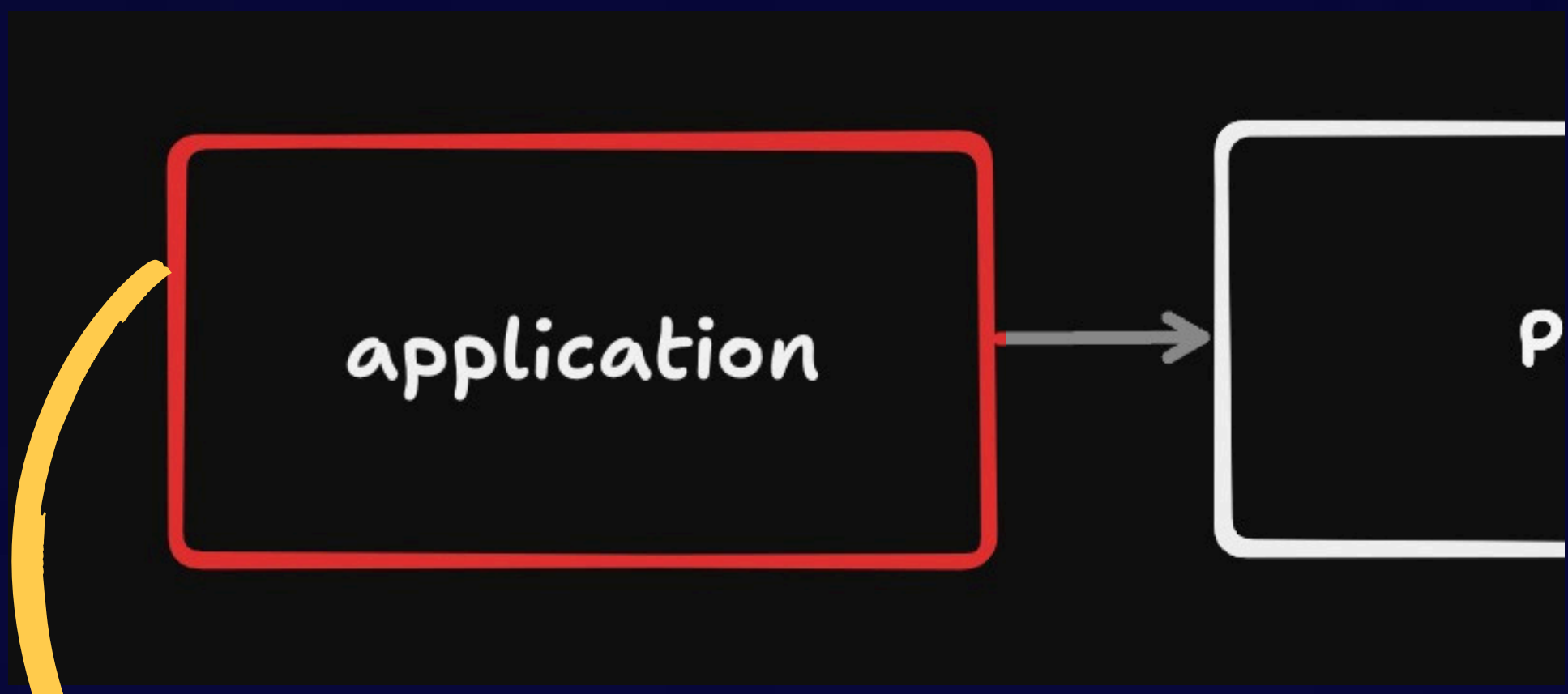
Isaac Gomes

o que é arquitetura de software?

arquitetura de software remete a como vamos dividir nossas camadas, como elas iram se comunicar, como sera organizado, nossos Design de código..

Isaac Gomes

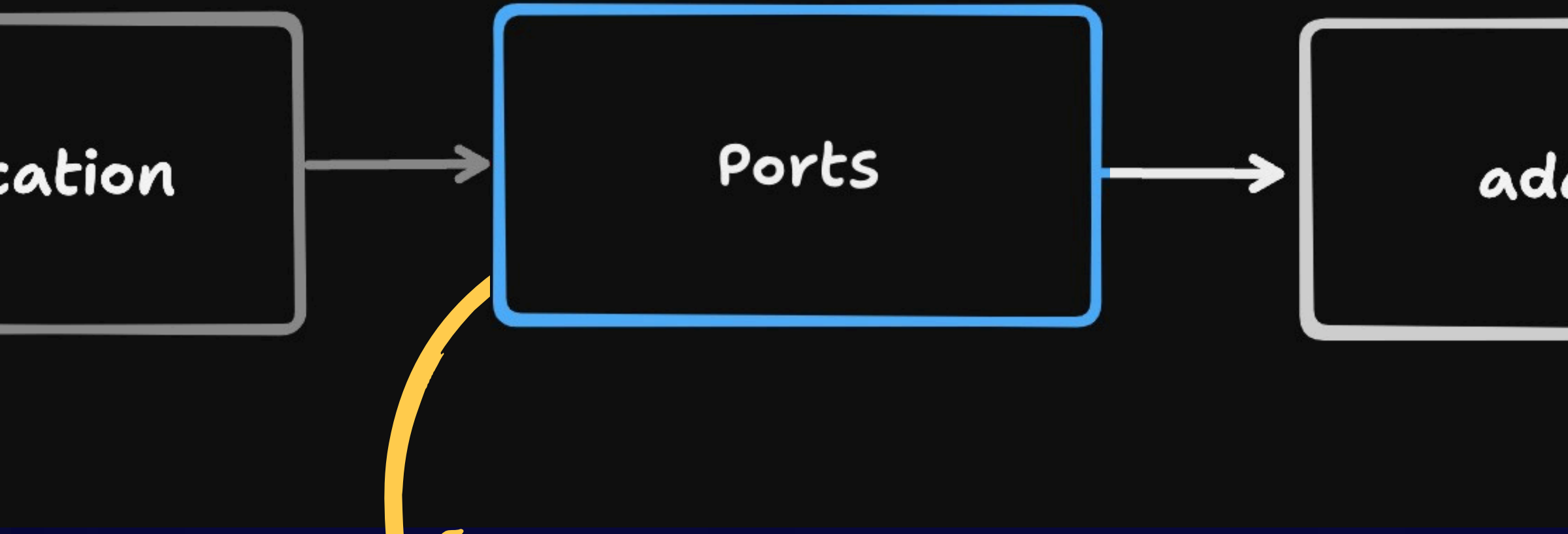
arquitetura hexagonal



Contém os casos de uso que orquestram a lógica de negócios

Isaac Gomes

arquitetura hexagonal

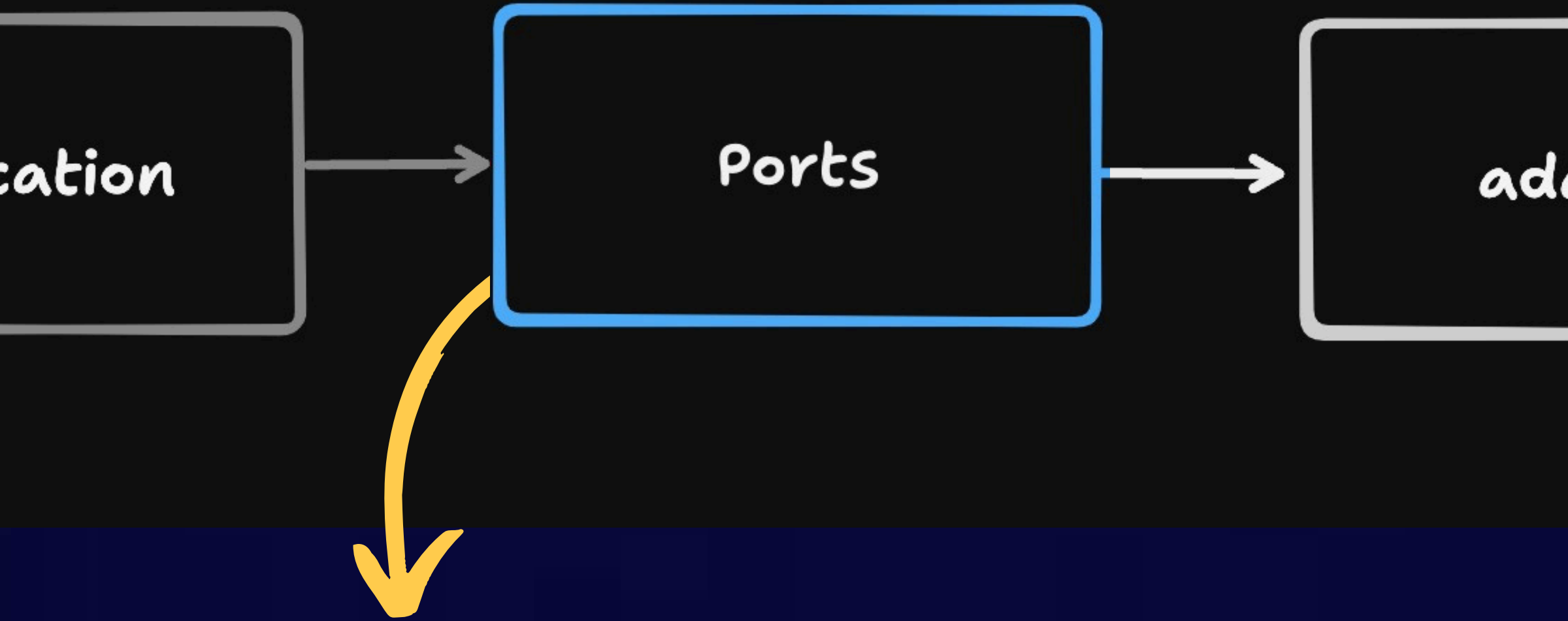


Inbound Ports

Estas portas representam casos de uso ou serviços que podem ser invocados por atores externos, como controladores de API ou interfaces de usuário.

Isaac Gomes

arquitetura hexagonal



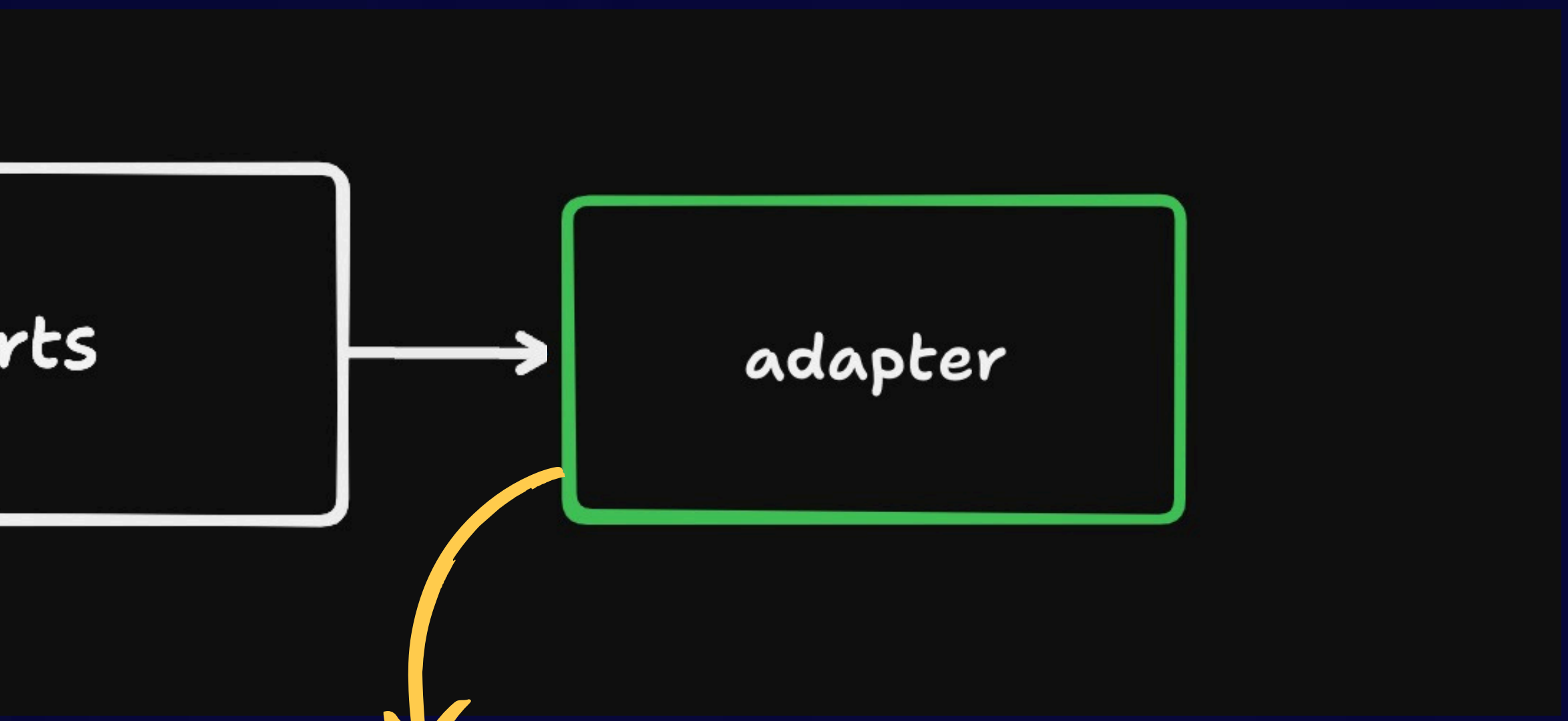
Outbound Ports

Estas portas representam operações que o núcleo da aplicação pode realizar em componentes externos, como repositórios, serviços de terceiros ou adaptadores de infraestrutura.

Isaac Gomes

TS

arquitetura hexagonal



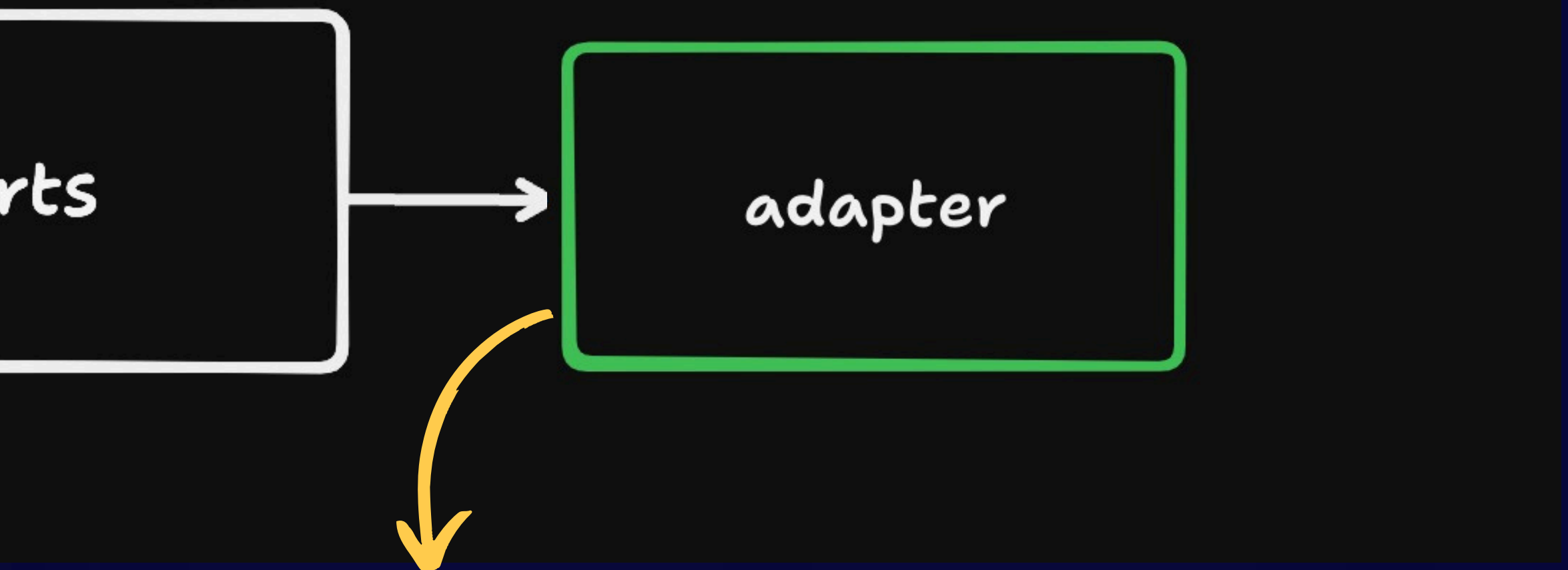
Inbound Adapters

São responsáveis por adaptar as interações de entrada para o núcleo da aplicação. Esses adaptadores recebem solicitações externas (por exemplo, requisições HTTP) e as transformam em chamadas para os casos de uso definidos pelas portas de entrada.

Isaac Gomes

TS

arquitetura hexagonal



Outbound Adapters

São responsáveis por adaptar as interações de saída do núcleo da aplicação para sistemas externos. Eles recebem chamadas do núcleo (através das portas de saída) e as transformam em operações específicas de infraestrutura (como salvar dados em um banco de dados).

Isaac Gomes

TS

arquitetura hexagonal

como seria um exemplo pratico disso?

vamos criar um cenário para exemplificar onde precisamos criar um novo usuário e fazer uma busca por id

Isaac Gomes

1 - definir como será nossa entidade de User no dominio

```ts
export class User {
  public id: string
  public name: string
  public email: string

  constructor({ name, email, id }: IUser) {
    this.id = this.name = id
    this.email = email
    this.name = name
  }
}
```

Isaac Gomes

TS

2 - criar nosso application

```
interface UserServicePort {
  createUser(dataUser: IUser): User
  getUser(id: string): User
}

class UserService implements UserServicePort {
  constructor() {}
  createUser(data: IUser): User {}
  getUser(id: string): User {}
}
```

aqui é onde acontecera a orquestração das nossas abstrações

Isaac Gomes

2 - criar nosso application

```
class UserService implements UserServicePort {
  constructor() {}
  createUser(data: IUser): User {}
  getUser(id: string): User {}
}
```

aqui vamos precisar realizar a persistência dos dados

```
export interface UserRepository {
  save(user: User): User
  findById(id: string): User
}
```

então criaremos uma Port para acessar essa persistência... veja não estamos falando de banco e sim de contrato

Isaac Gomes

TS

3 - criar nosso adapter

```ts
export class UserRepositoryMemoryAdapter implements UserRepository {
  private users: Map<string, User> = new Map()

  save(user: User): User {
    this.users.set(user.id, user)
    return user
  }

  findById(id: string): User {
    const user = this.users.get(id)
    if (!user) throw new Error(`User with id ${id} not found`)
    return user
  }
}
```

criei um repository para simular um teste de persistência... se trabalha com testes provavelmente ja viu uma das utilidades

Isaac Gomes

4 - adicionar o contrato da porta e injetar o adapter no nosso adapter

```ts
export class UserService implements UserServicePort {
  constructor(
    private userRepository: UserRepository,
  ) {}

  createUser(data: IUser): User {}
  getUser(id: string): User {}
}
```

agora temos o nosso modo de persistência disponível na nossa application pronto para ser orquestrado

Isaac Gomes

5 - imprementar a criação e getUser

```ts
export class UserService implements UserServicePort {
  constructor(private userRepository: UserRepository) {}

  createUser(data: IUser): User {
    const user = new User(data)
    return this.userRepository.save(user)
  }

  getUser(id: string): User {
    return this.userRepository.findById(id)
  }
}
```

Isaac Gomes

6 - agora é só fazer o uso

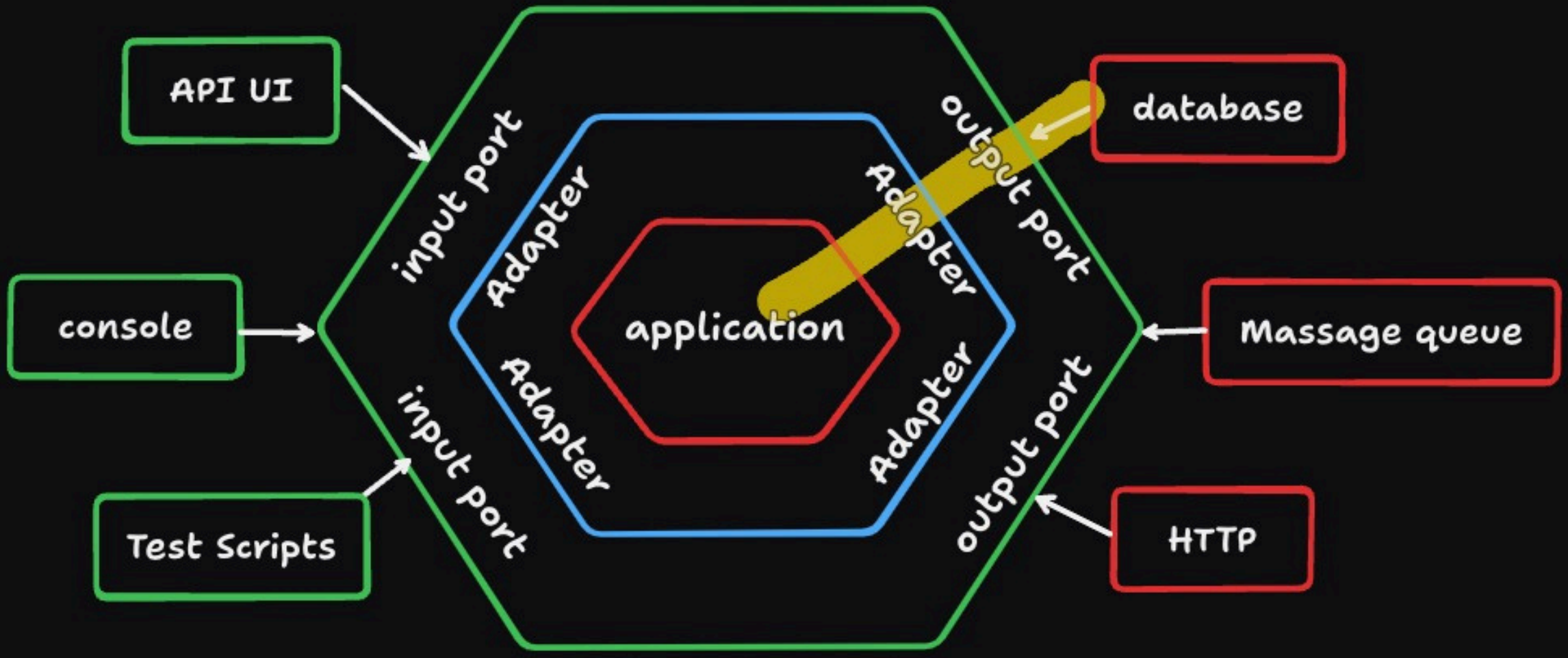
```ts
const userRepository = new UserRepositoryMemoryAdapter()
const userService = new UserService(userRepository)

userService.createUser({
  email: 'john@example.com',
  name: 'John Doe',
  id: '1',
})

const user = userService.getUser('1')
```

obvio que em uma aplicação de verdade vc não passa o id...
foi só para exemplificar

Isaac Gomes

TS

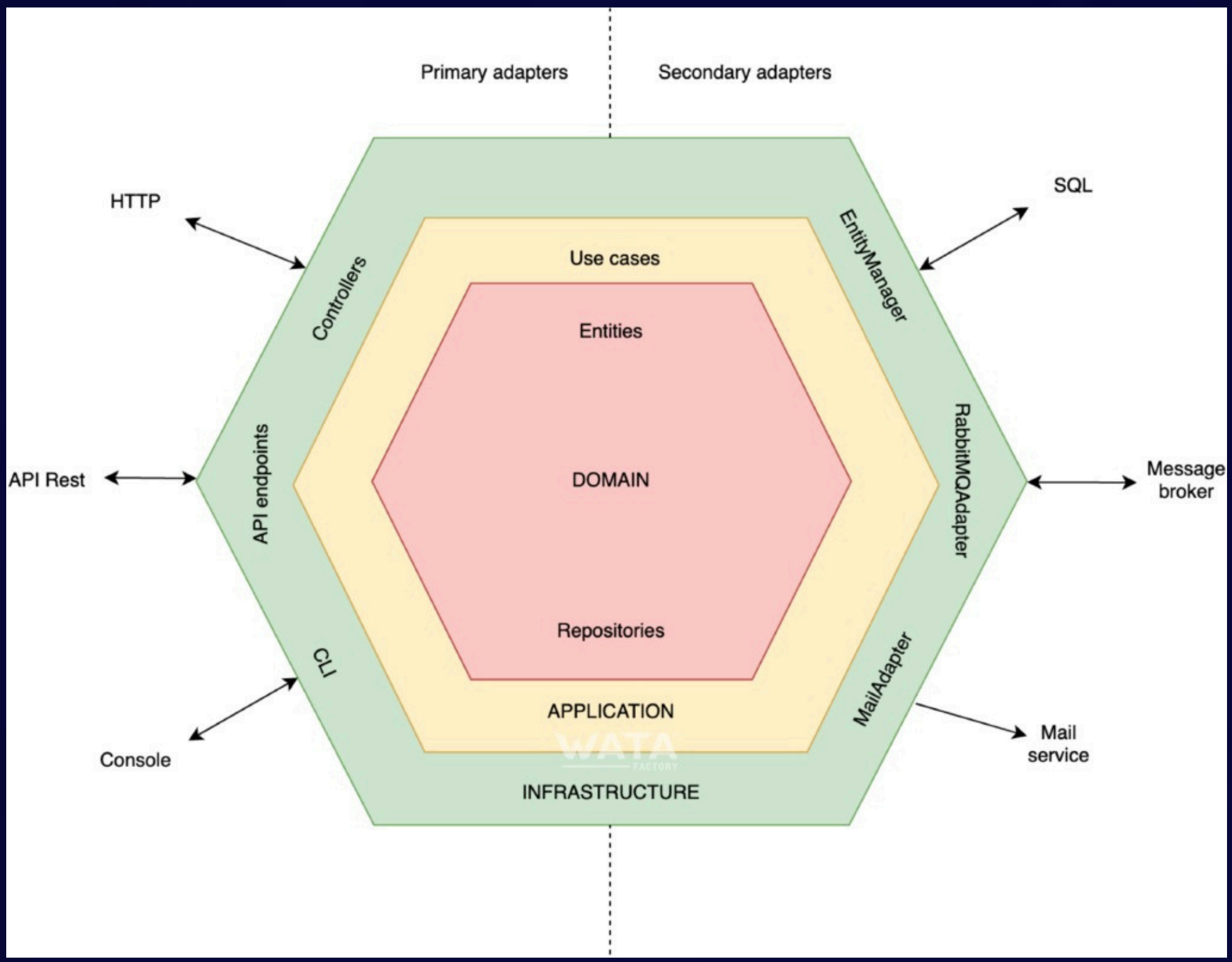
- *agora as peças fazem mais sentido*



Isaac Gomes

TS

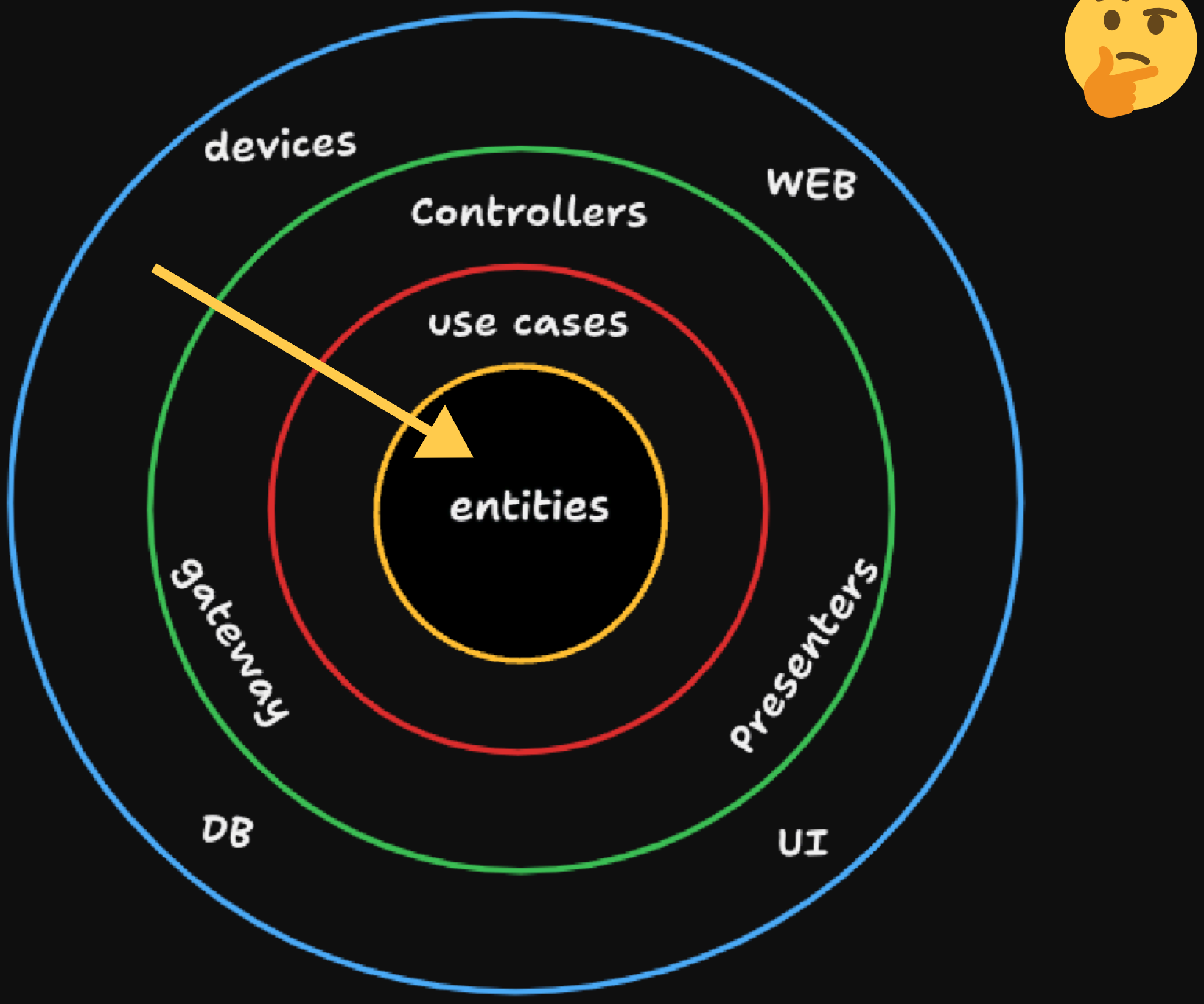
- ***estrutura de pastas comum***



os nomes mais comuns são os acima, porem, vale lembrar que não tem uma nomenclatura obrigatória

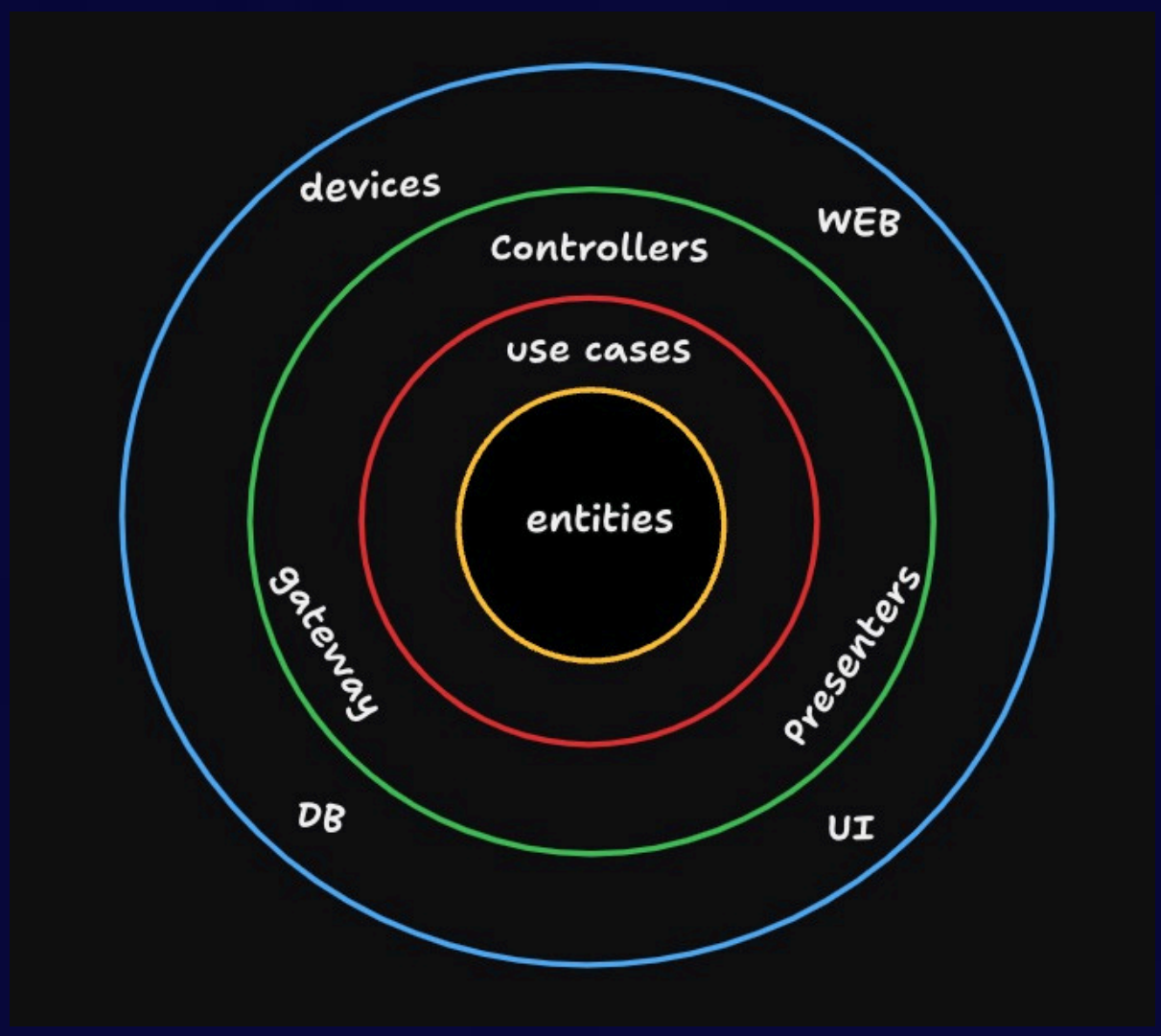
Isaac Gomes

Desvendando
Clean Architecture



Isaac Gomes

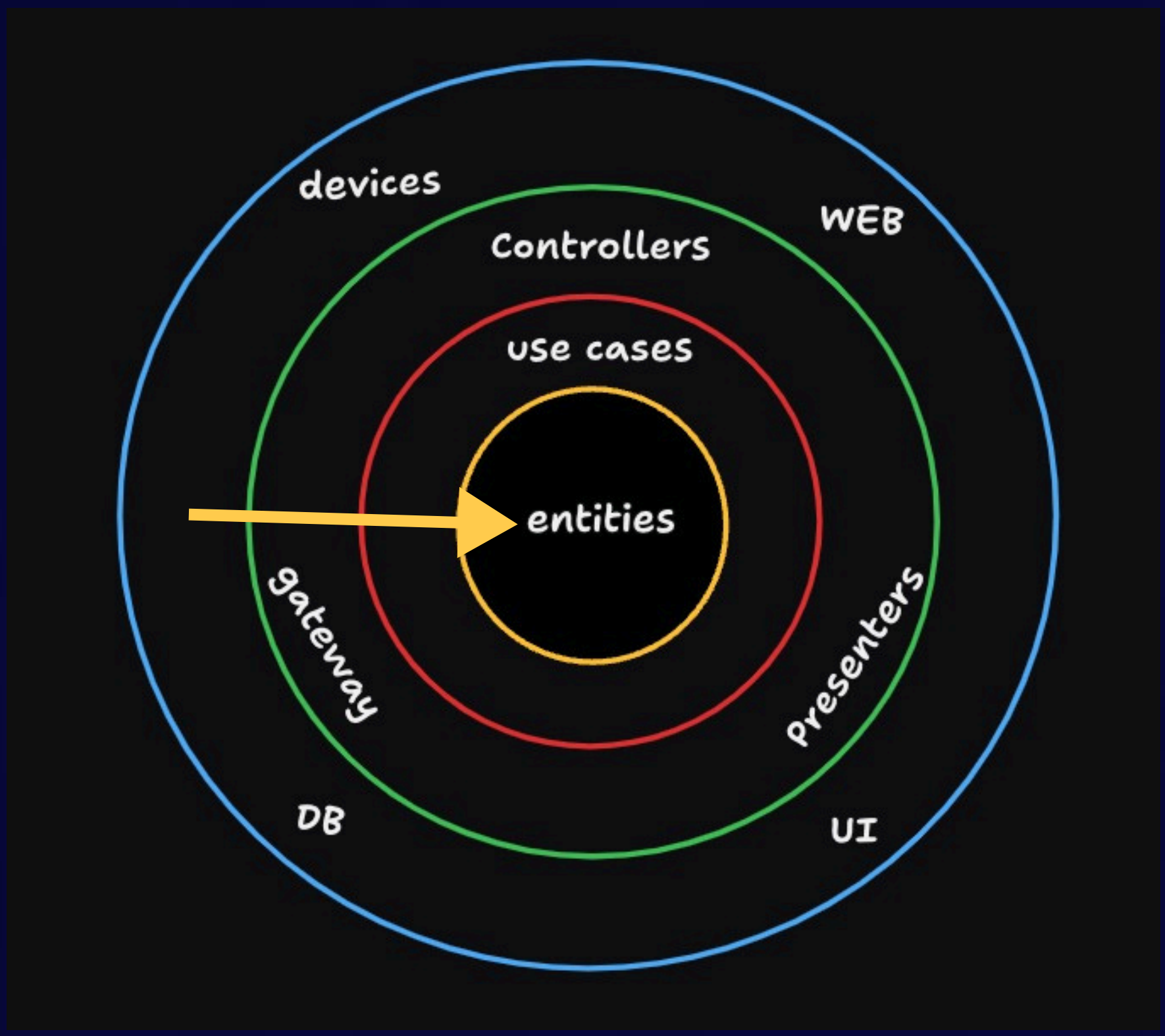
o que é Clean Architecture ?



Clean Architecture é um padrão de design de software que organiza o código em camadas independentes, facilitando a manutenção e escalabilidade. Ele separa a lógica de negócios da infraestrutura e da interface, promovendo uma estrutura modular e testável.

Isaac Gomes

o que é Clean Architecture ?



o fluxo de consumo de baseia de **fora para dentro** de uma maneira que temos uma **independência de camadas**

Isaac Gomes

o que é Entities ?

- Contém as **regras de negócio** mais genéricas e independentes

- Não depende de nenhuma outra camada.

Isaac Gomes

o que é useCases ?

- Contém a **lógica específica** da aplicação, ou seja, os fluxos de trabalho do sistema.

-São os serviços que coordenam as interações entre as **entidades**.

-Depende apenas das **entidades**.

Isaac Gomes

o que é Controllers ?

-Contém os **adaptadores** e controladores que convertem dados entre o formato interno das camadas Use **Cases/Entities** e o formato externo da camada **Frameworks & Drivers**.

-Exemplos incluem **controladores de API**, **repositórios**, e **transformadores de dados**.

Isaac Gomes

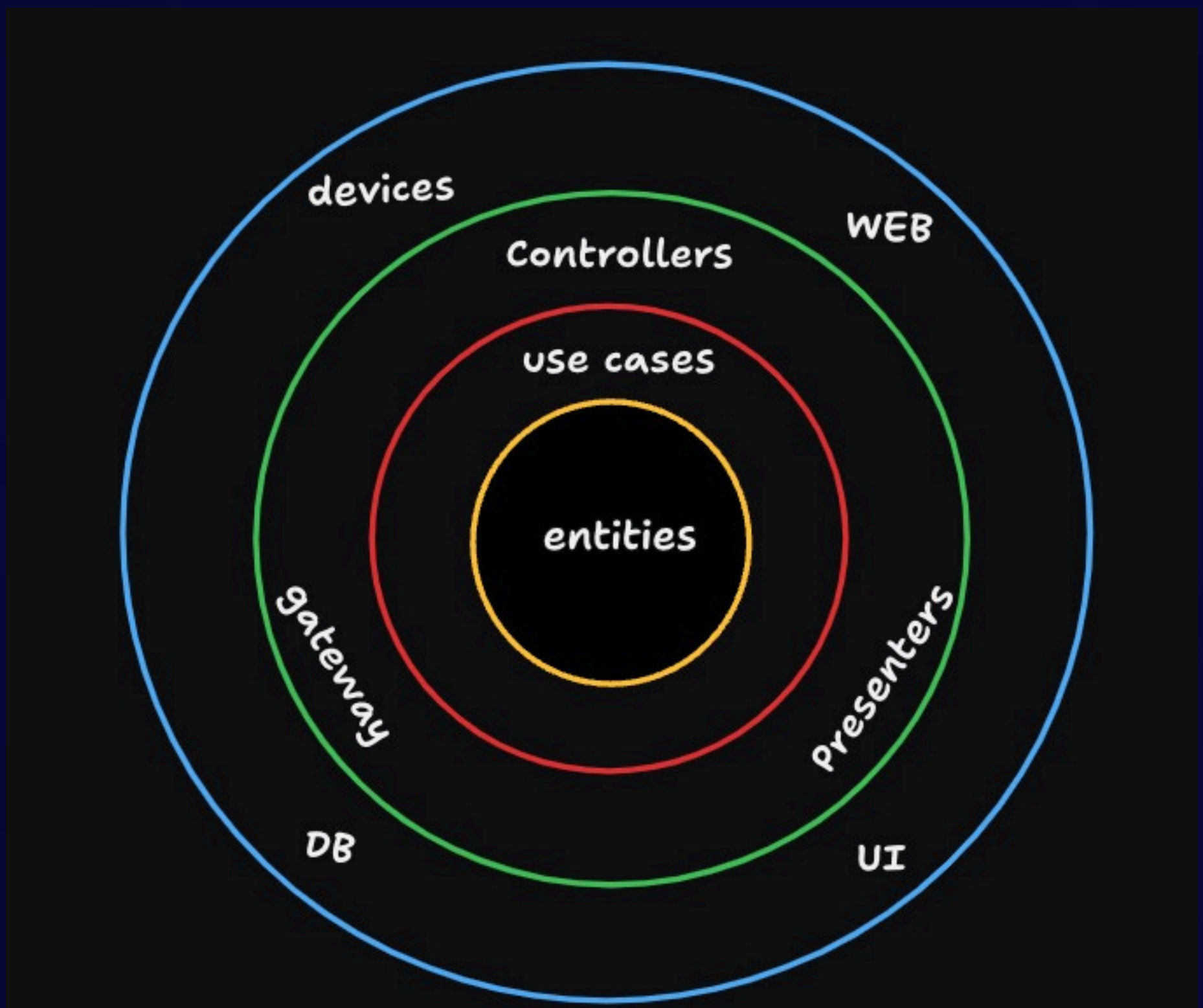
o que é Frameworks?

-Contém a implementação de detalhes como **banco de dados**, **frameworks da web**, etc.

-Esta camada é a mais externa e pode ser **facilmente trocada**.

Isaac Gomes

agora que sabemos o que cada camada representa



como fica a implementação disso na pratica?

Isaac Gomes

cenário

devemos criar dois recursos
de uma lista de tarefa

cadastrar uma nova tarefa
Listar todas as tarefas

Isaac Gomes

1- criar nossa entidade da Task

```ts
export class Task implements ITask {
  public id: string
  public title: string
  public description: string

  constructor({ id, title, description }: ITask) {
    this.id = id
    this.title = title
    this.description = description
  }
}
```

Isaac Gomes

TS

2 - criar nosso useCase

```ts
export class CreateTask {
  constructor(private taskRepository: TaskRepository) {}

  execute(title: string, description: string): Task {
    const task = new Task({
      id: generateId(),
      title,
      description,
    })
    this.taskRepository.save(task)
    return task
  }
}
```

Isaac Gomes

3 - vamos criar nosso TaskRepository

```
export class InMemoryTaskRepository implements TaskRepository {
  private tasks: Task[] = []

  save(task: Task): void {
    this.tasks.push(task)
  }

  findAll(): Task[] {
    return this.tasks
  }
}
```

Isaac Gomes

4 - vamos criar nosso TaskController

```
export class TaskController {
  private createTask: CreateTask
  private taskRepository: TaskRepository

  constructor(dependencies: TaskControllerDependencies) {
    this.createTask = dependencies.createTask
    this.taskRepository = dependencies.taskRepository
  }

  create(req: Request, res: Response) {
    const { title, description } = req.body
    const task = this.createTask.execute(title, description)
    res.json(task)
  }

  findAll(_req: Request, res: Response) {
    const tasks = this.taskRepository.findAll()
    res.json(tasks)
  }
}
```

Isaac Gomes

TS

5 - agora vamos expor isso no nosso framework

```
const app = express()
app.use(express.json())

const taskRepository = new InMemoryTaskRepository()
const createTask = new CreateTask(taskRepository)
const taskController = new TaskController({
  createTask,
  taskRepository,
})

app.post('/tasks', (req, res) => taskController.create(req, res))
app.get('/tasks', (req, res) => taskController.findAll(req, res))

app.listen(3000, () => console.log('Server is running on port 3000'))
```

Isaac Gomes

TS

6 - agora é só testar {create}

POST http://localhost:3000/tasks

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
  "title": "Sample Task",
  "description": "This is a description of the sample task."
}
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "id": "ud9ggdcd35",
    "title": "Sample Task",
    "description": "This is a description of the sample task."
}
```

Isaac Gomes

TS

6 - agora é só testar {getAll}

GET http://localhost:3000/tasks

Params Authorization Headers (7) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1
```

Body Cookies Headers (7) Test Results

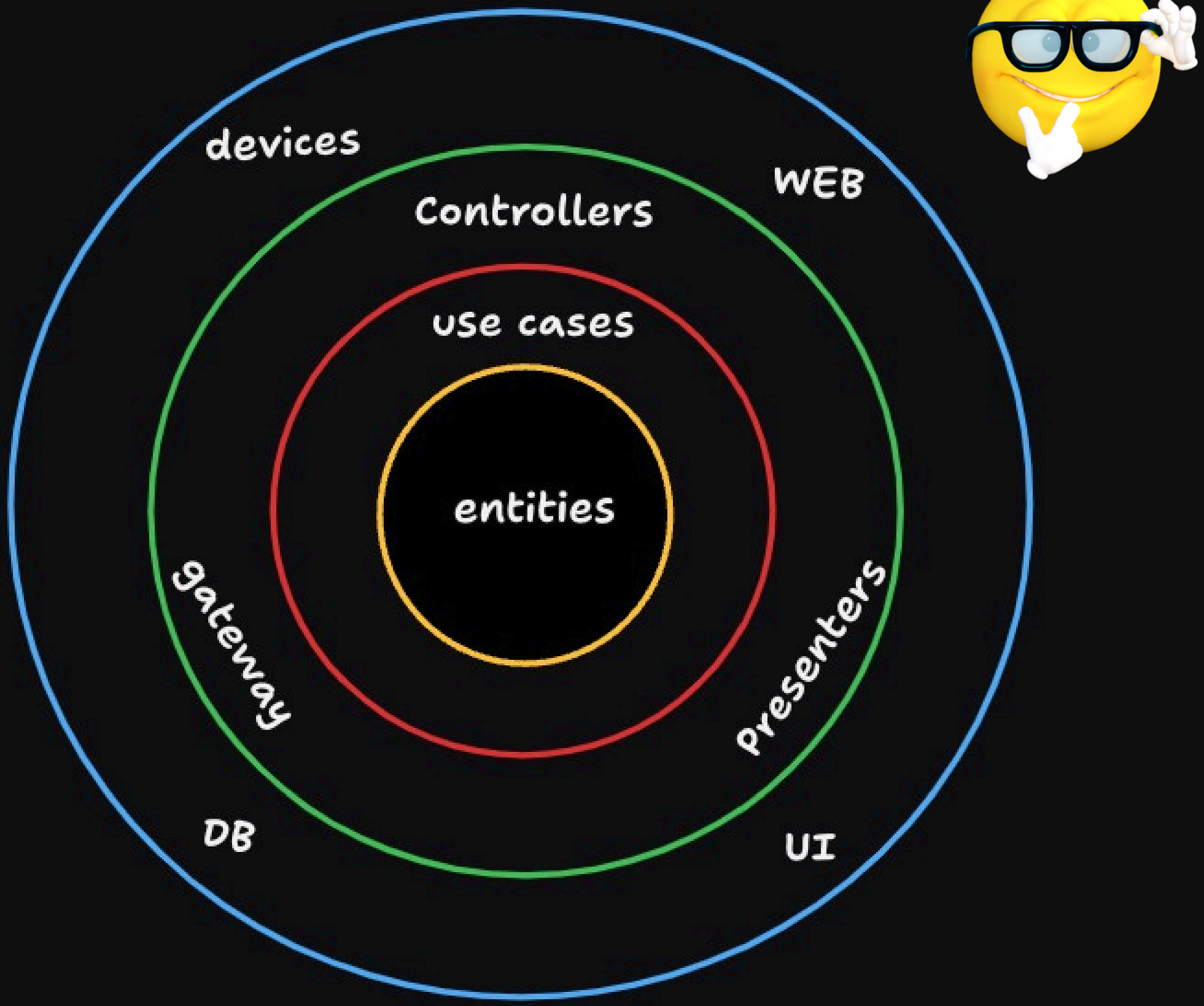
Pretty Raw Preview Visualize JSON

```
1  [
2      {
3          "id": "ud9ggdcd35",
4          "title": "Sample Task",
5          "description": "This is a description of the sample task."
6      }
7  ]
```

Isaac Gomes

TS

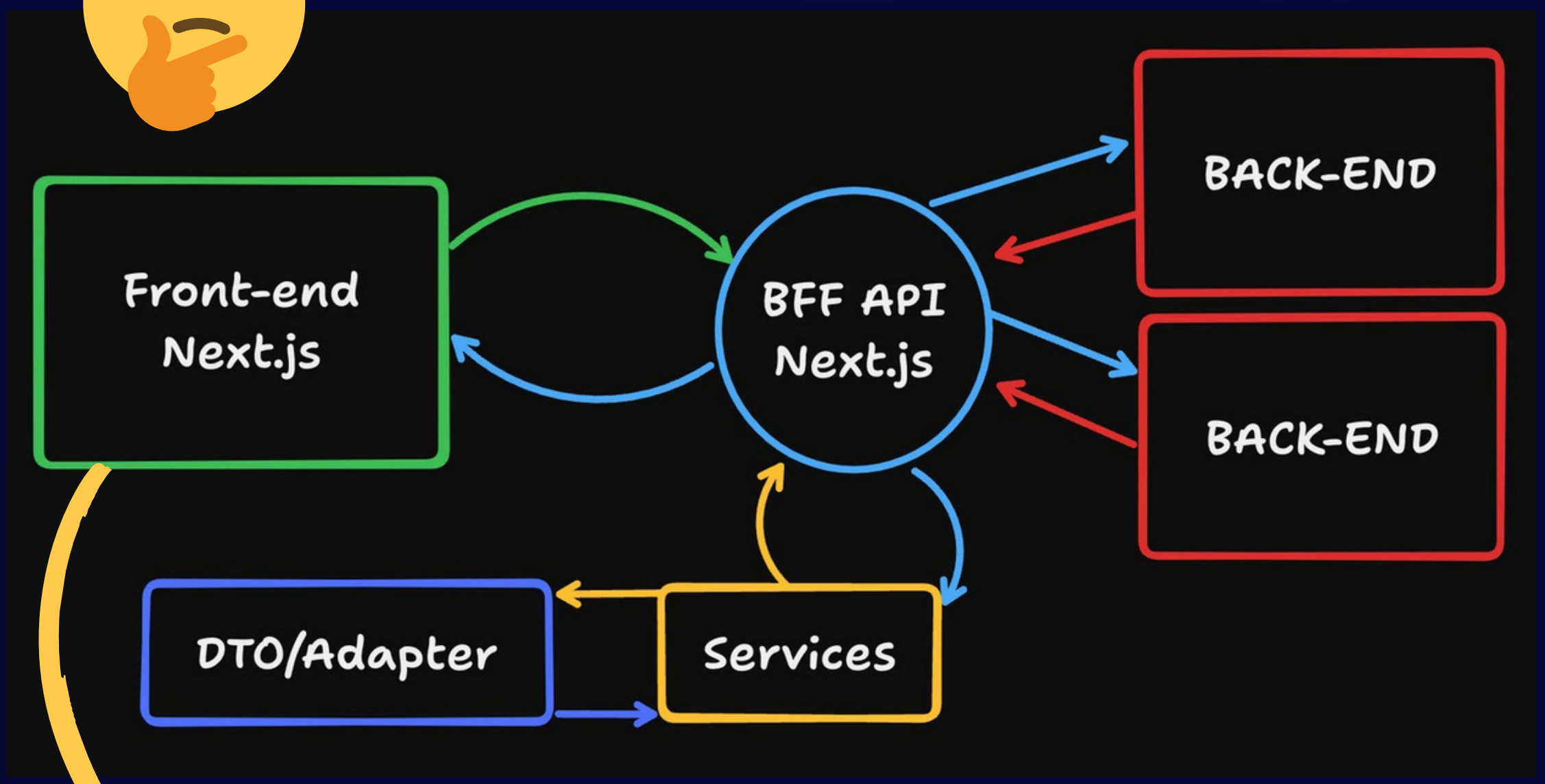
agora as coisas começam a tomar forma



Isaac Gomes

TS

arquitetura baseada em BFF (Backend for Frontend)



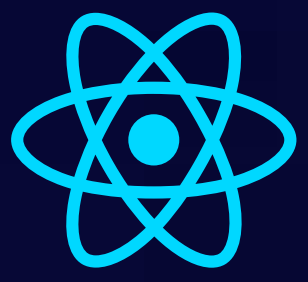
exemplo em Next.js

Isaac Gomes

o que é
Backend for Frontend
???

Backend for Frontend (BFF) é um padrão de arquitetura usado para melhorar a eficiência e a experiência do usuário em aplicativos Front-end.

Isaac Gomes

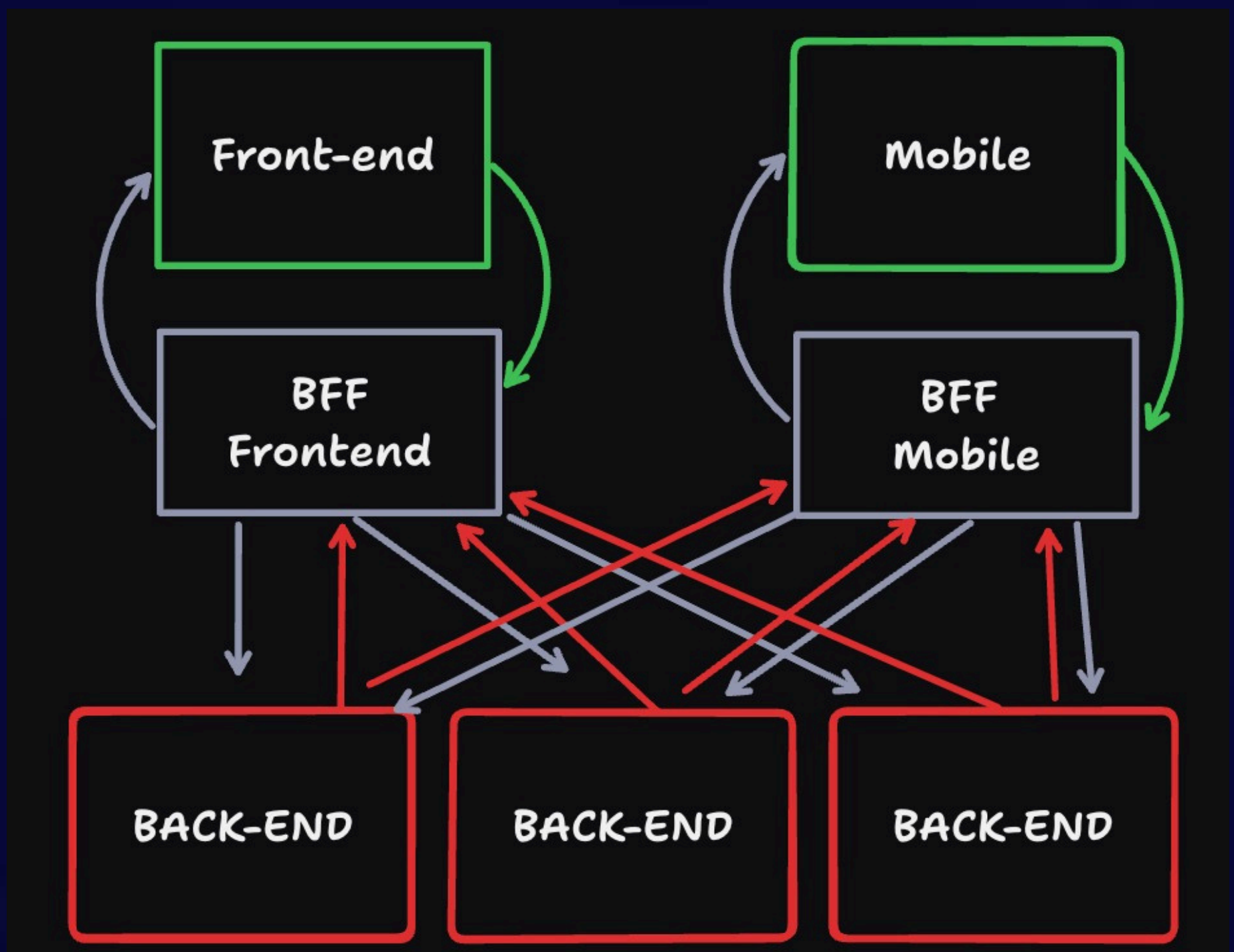


para que serve
Backend for Frontend
???

A ideia principal por trás do BFF é criar um backend específico para cada tipo de cliente (por exemplo, aplicativos móveis, web, etc.) em vez de um único backend genérico para todos os tipos de clientes.

Isaac Gomes

exemplo de como **Backend for Frontend** funciona



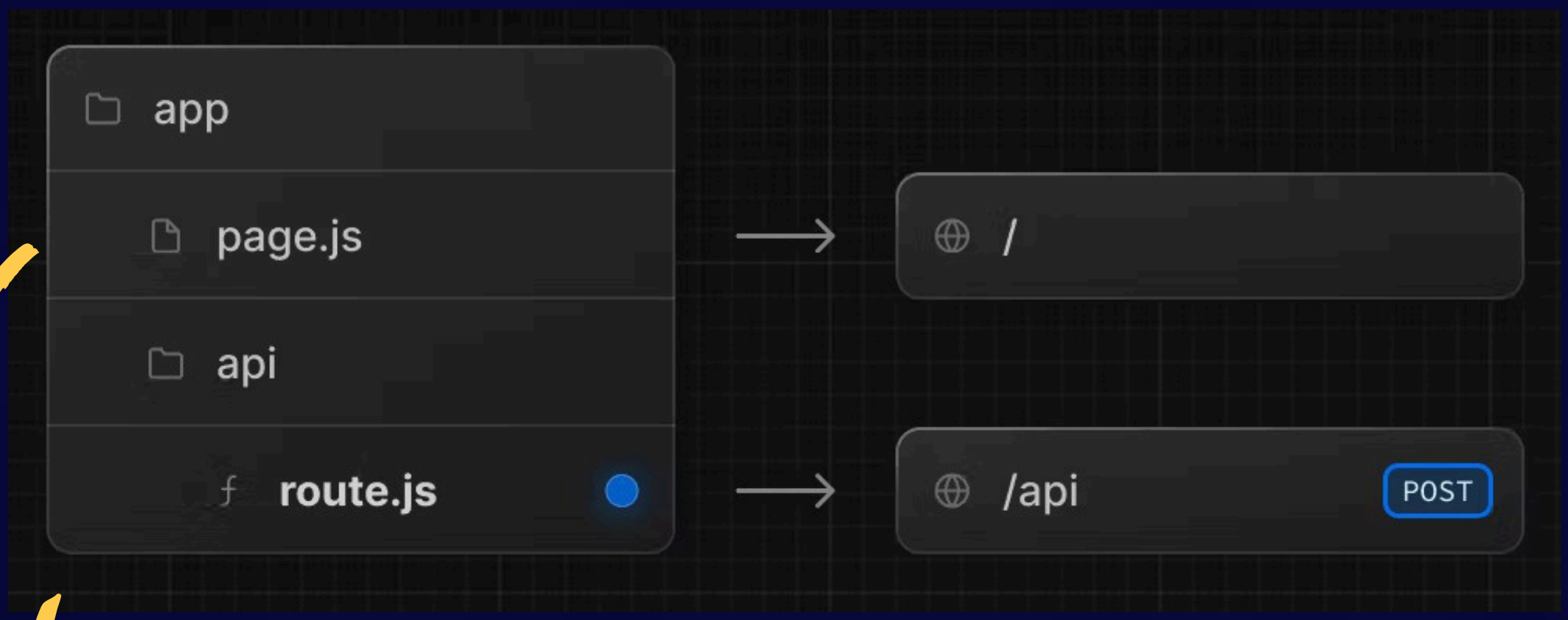
Isaac Gomes

Beneficios do BFF

- **Customização**
- **Desempenho Otimizado**
- **Segurança**
- **Desacoplamento**

Isaac Gomes

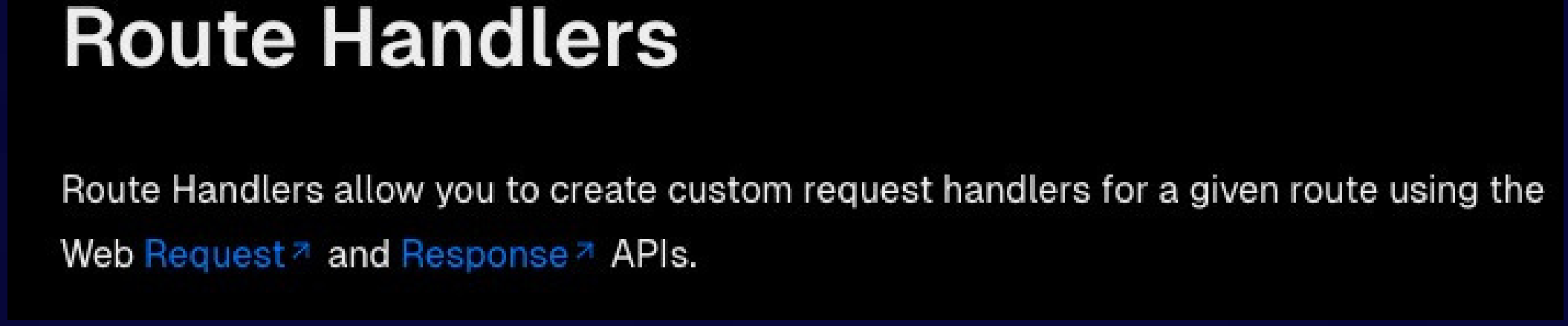
Como Podemos Aplicar BFF em Next.js ???



Next.js é um Framework Fullstack conseguimos criar rotas backend dentro da pasta API

Isaac Gomes

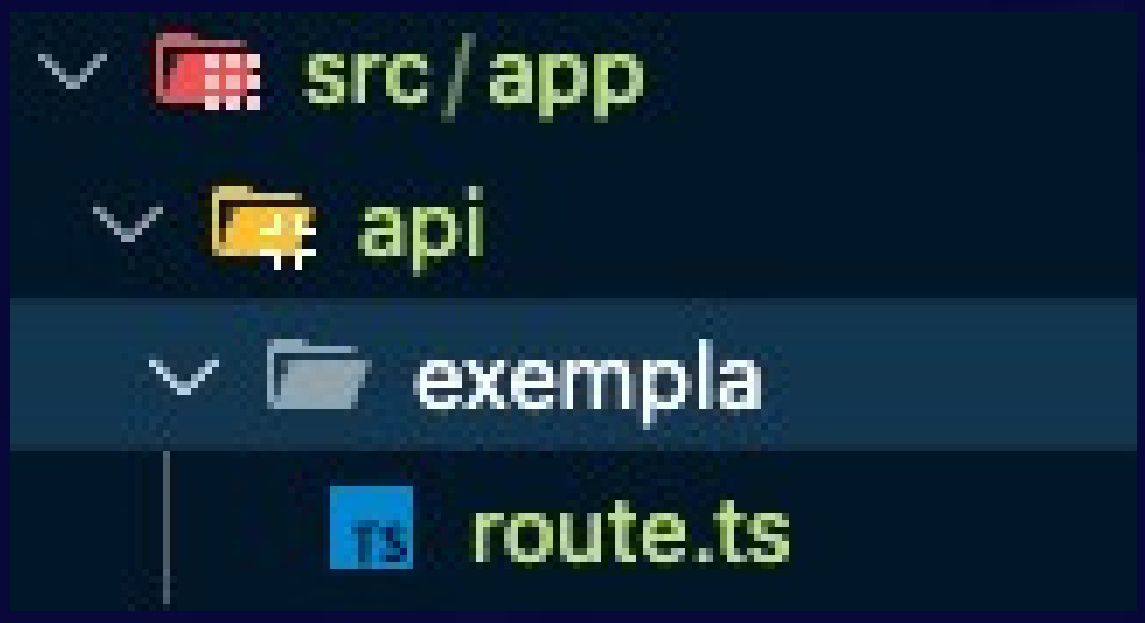
Onde acho informações sobre a criação de rotas na API ???



basta pesquisar na documentação por Route Handlers

Isaac Gomes

como criar um rota na API do Next.js



para criar um rota vc precisa criar a pasta API dentro da pasta APP e depois criar um arquivo com o nome que deseja e criar o file com o nome route

Isaac Gomes

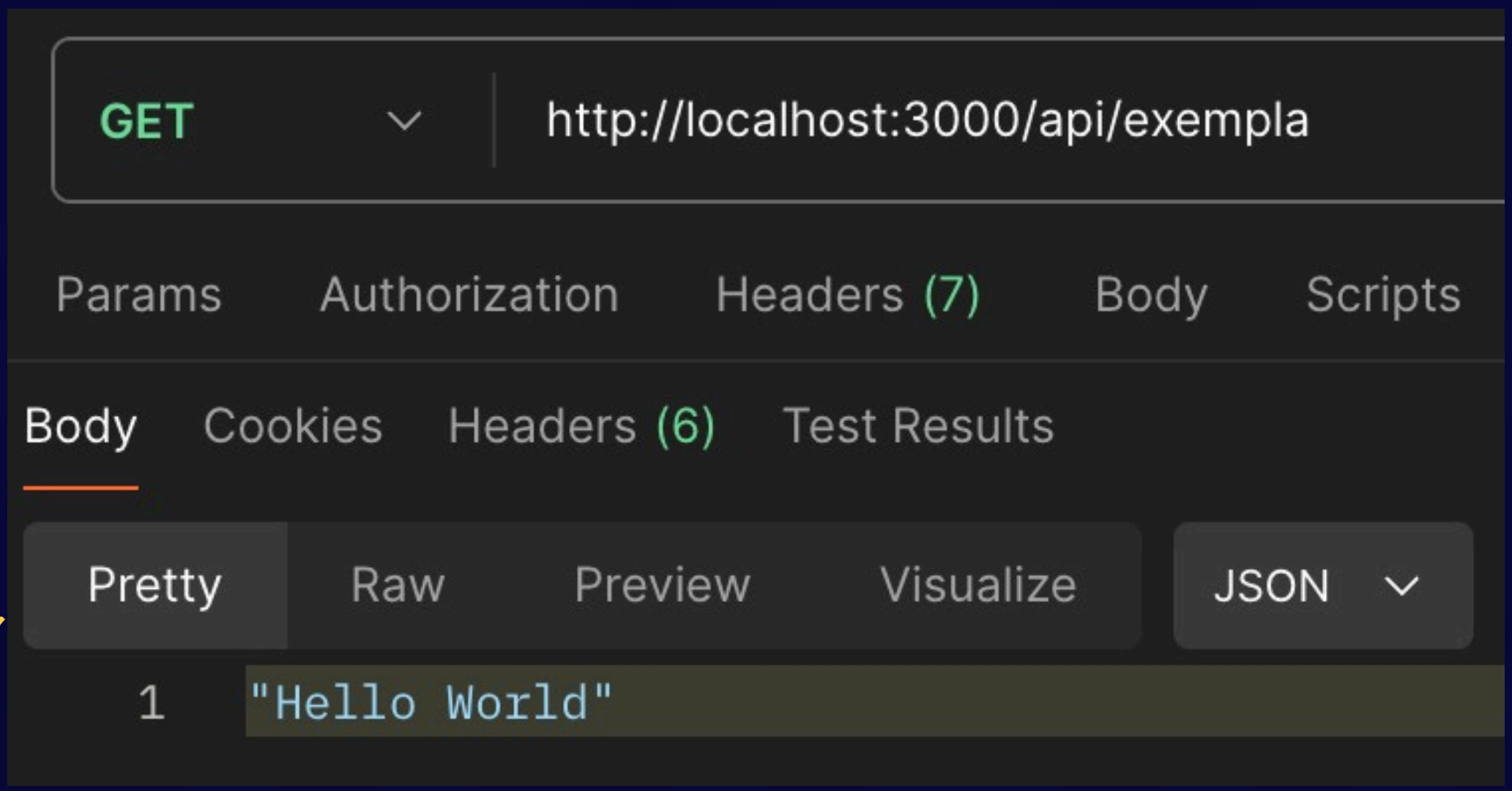
como criar um rota na API do Next.js

```
export async function GET(request: Request) {
  return NextResponse.json(
    'Hello World', {
    status: 200,
  })
}
```

agora vamos criar a rota de acesso no caso quero um GET então para acessar basta url/api/nome_do_arquivo no type que definiu

Isaac Gomes

como criar um rota na API do Next.js



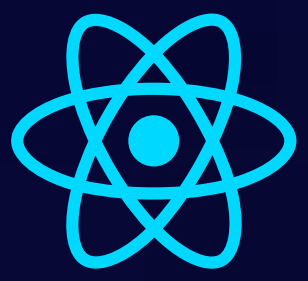
agora basta acessar sua rota

Isaac Gomes

exemplo real de rota em Next.js

consumir um endpoint da api do rick, um endpoint da api do json placeholder e realizar um mapeamento dos dados para estruturas que fazem sentido para o client

Isaac Gomes



Consultando API

```
export async function GET(req: Request) {
  try {
    const [posts, characters] = await Promise.all([
      fetchPosts(),
      fetchRickAndMortyCharacters()
    ])
    return NextResponse.json(
      adapterResponseGetPerson({
        posts,
        characters
      }),
      { status: 200 })
  } catch (error) {
    return NextResponse.json(
      'fetching posts and characters',
      { status: 404 })
  }
}
```

então aqui realizamos a consulta as duas API

Isaac Gomes

Adaptando a resposta

```
export async function GET(req: Request) {
  try {
    const [posts, characters] = await Promise.all([
      fetchPosts(),
      fetchRickAndMortyCharacters()
    ])
    return NextResponse.json(
      adapterResponseGetPerson({
        posts,
        characters
      }),
      { status: 200 })
  } catch (error) {
    return NextResponse.json(
      'fetching posts and characters',
      { status: 404 })
  }
}
```

aqui mapeamos os dados e retornamos

Isaac Gomes

consultando a resposta

GET http://localhost:3000/api/person

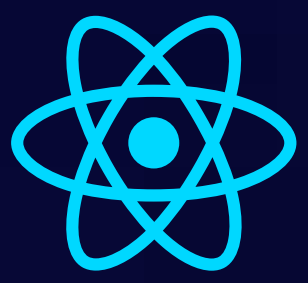
Params Authorization Headers (7) Body Scripts

ody Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON


```
{
    "characters": [ …
    ],
    "posts": [ …
    ]
}
```


Isaac Gomes



Request para nosso Route Handler

```
export async function getPostsandCharacters(){
  const { data } = await apiBFF.get('/person')
  return data
}

export default async function Home() {
  const data = await getPostsandCharacters()
  return <main>TESTE</main>
}
```

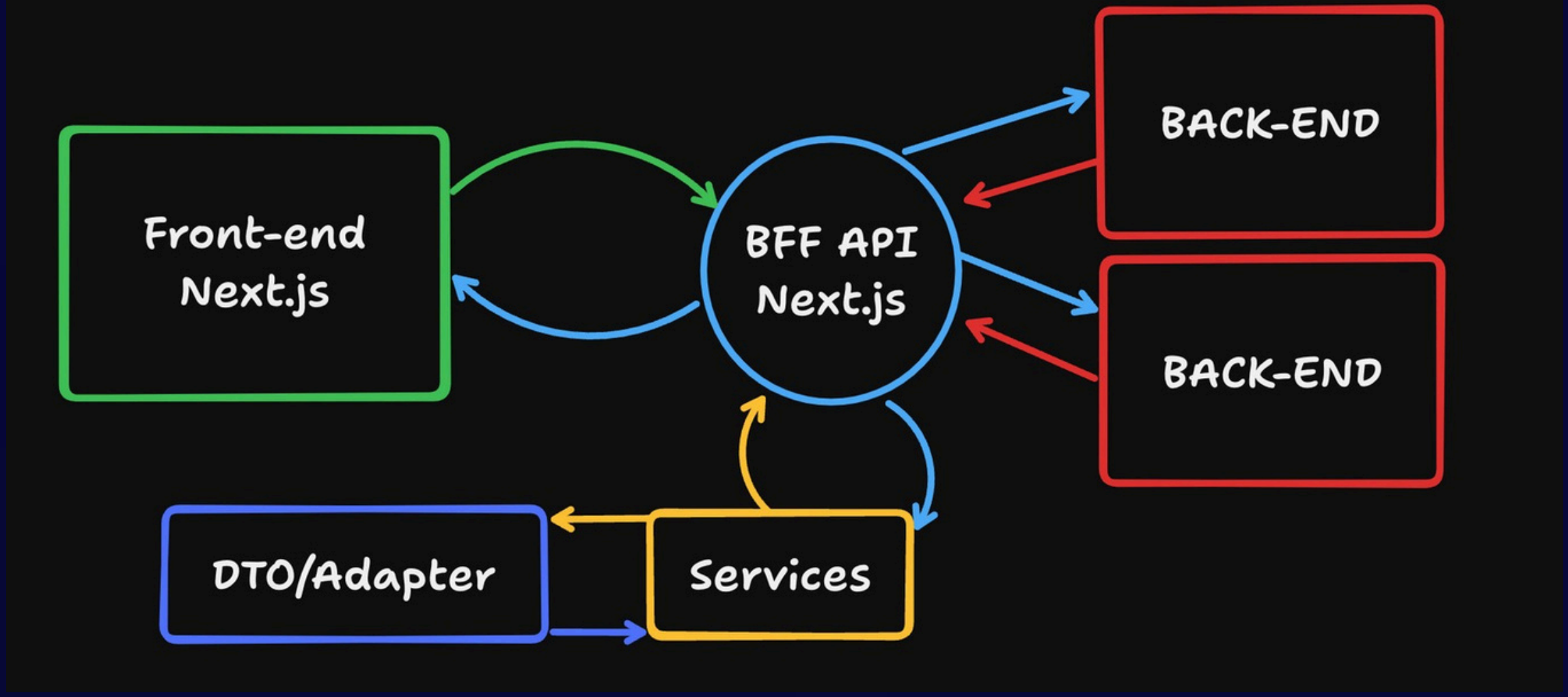
Isaac Gomes

Pronto estamos consumindo nossa rota


```
{
  characters: [
    {
      id: 1,
      name: 'Rick Sanchez',
      status: 'Alive',
      species: 'Human',
      type: ''
    },
    {
      id: 2,
      name: 'Morty Smith',
      status: 'Alive',
      species: 'Human',
      type: ''
    },
    {
      id: 3,
      name: 'Summer Smith',
      status: 'Alive',
      species: 'Human',
      type: ''
    },
    {
```

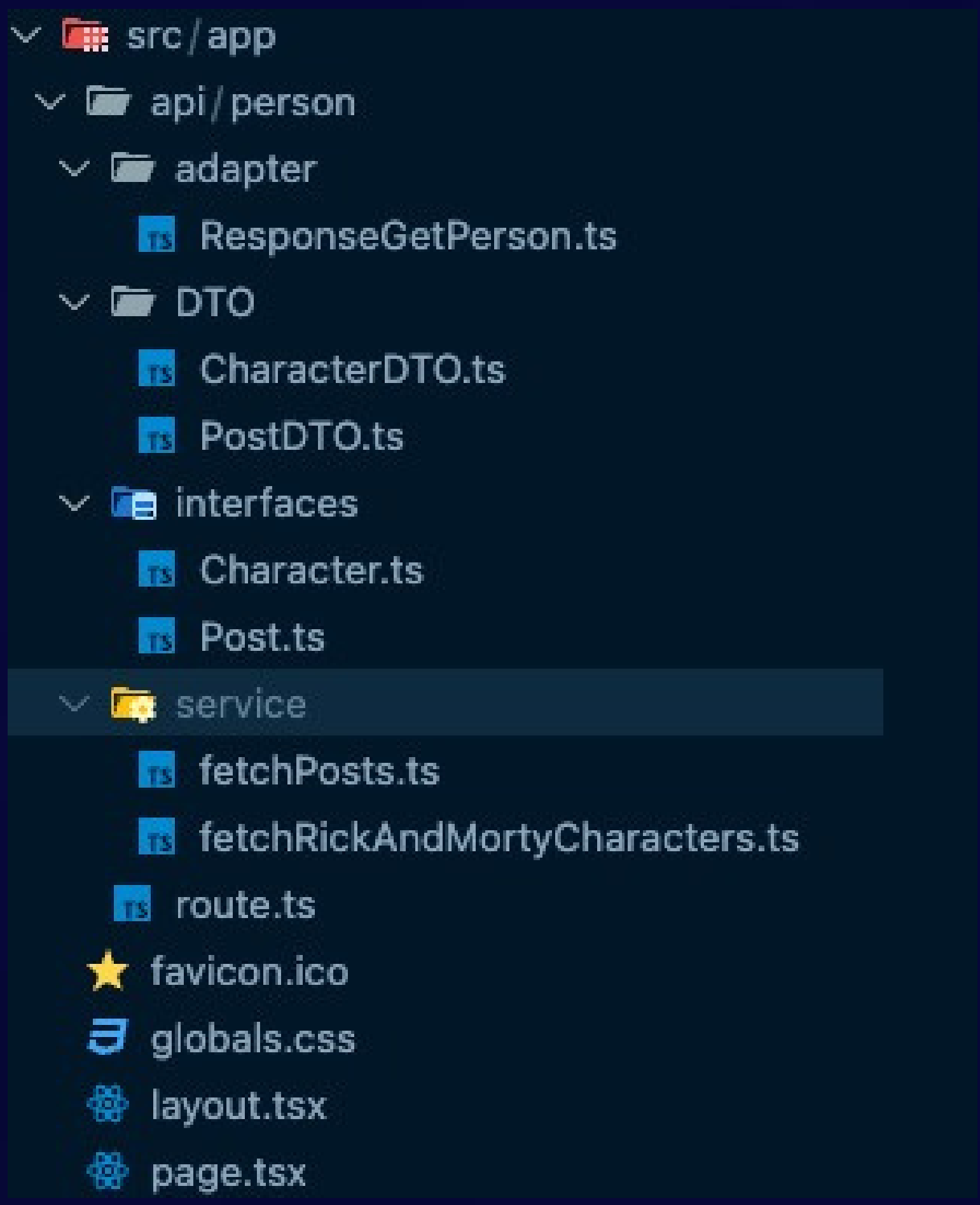

Isaac Gomes

aqui vemos o fluxo na pratica

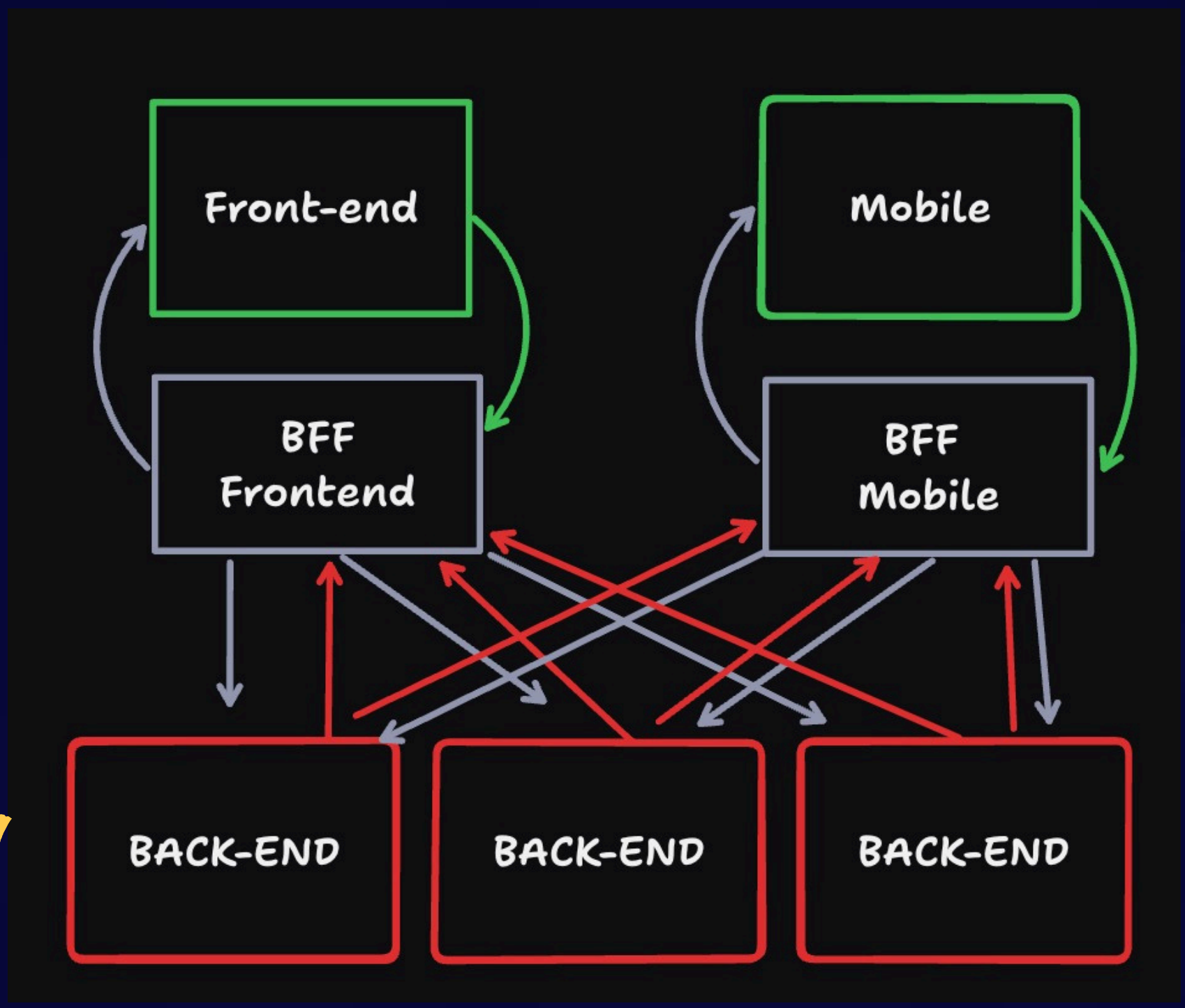


Isaac Gomes

como ficou a separação dos arquivos



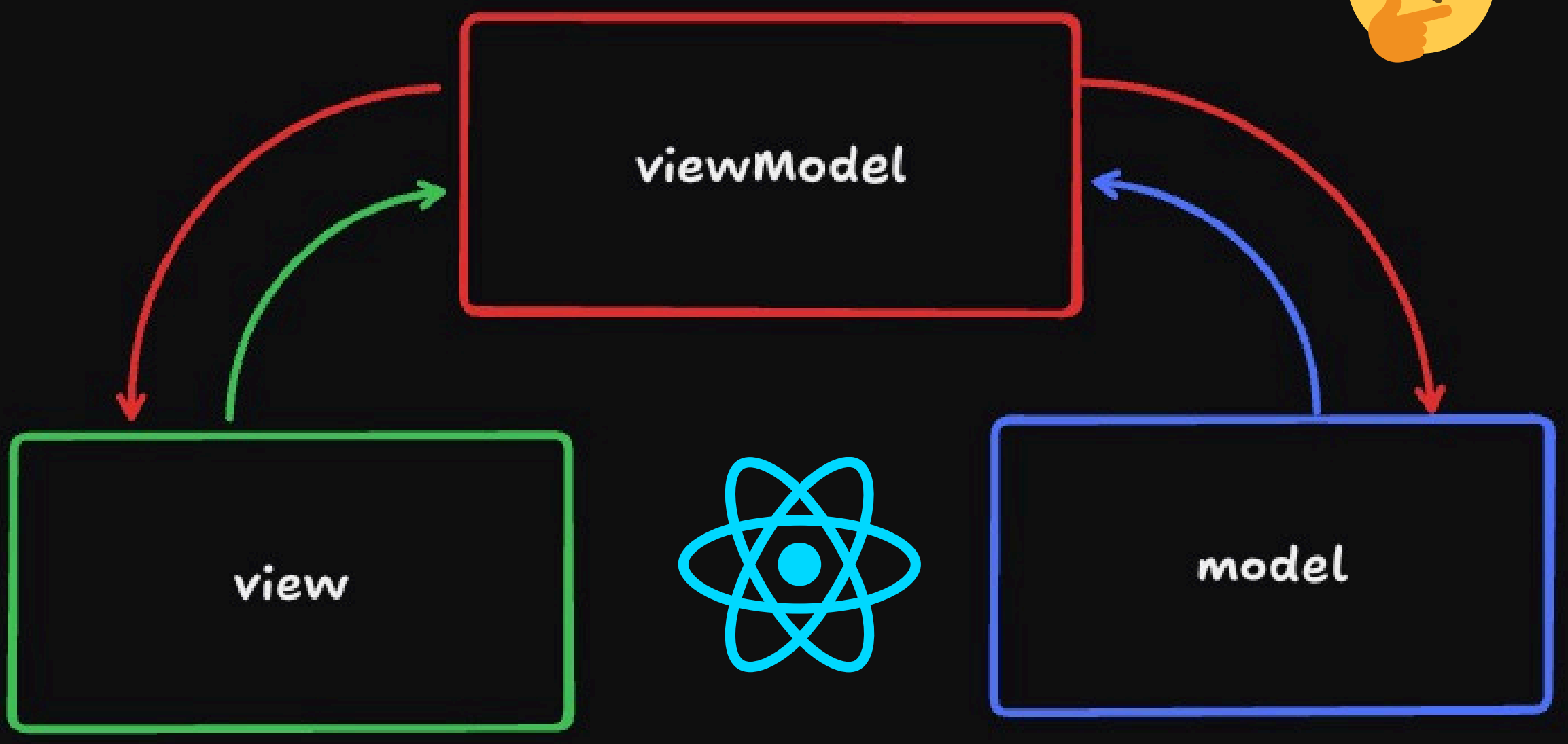
Isaac Gomes



agora essa imagem passa a ter mais significado

Isaac Gomes

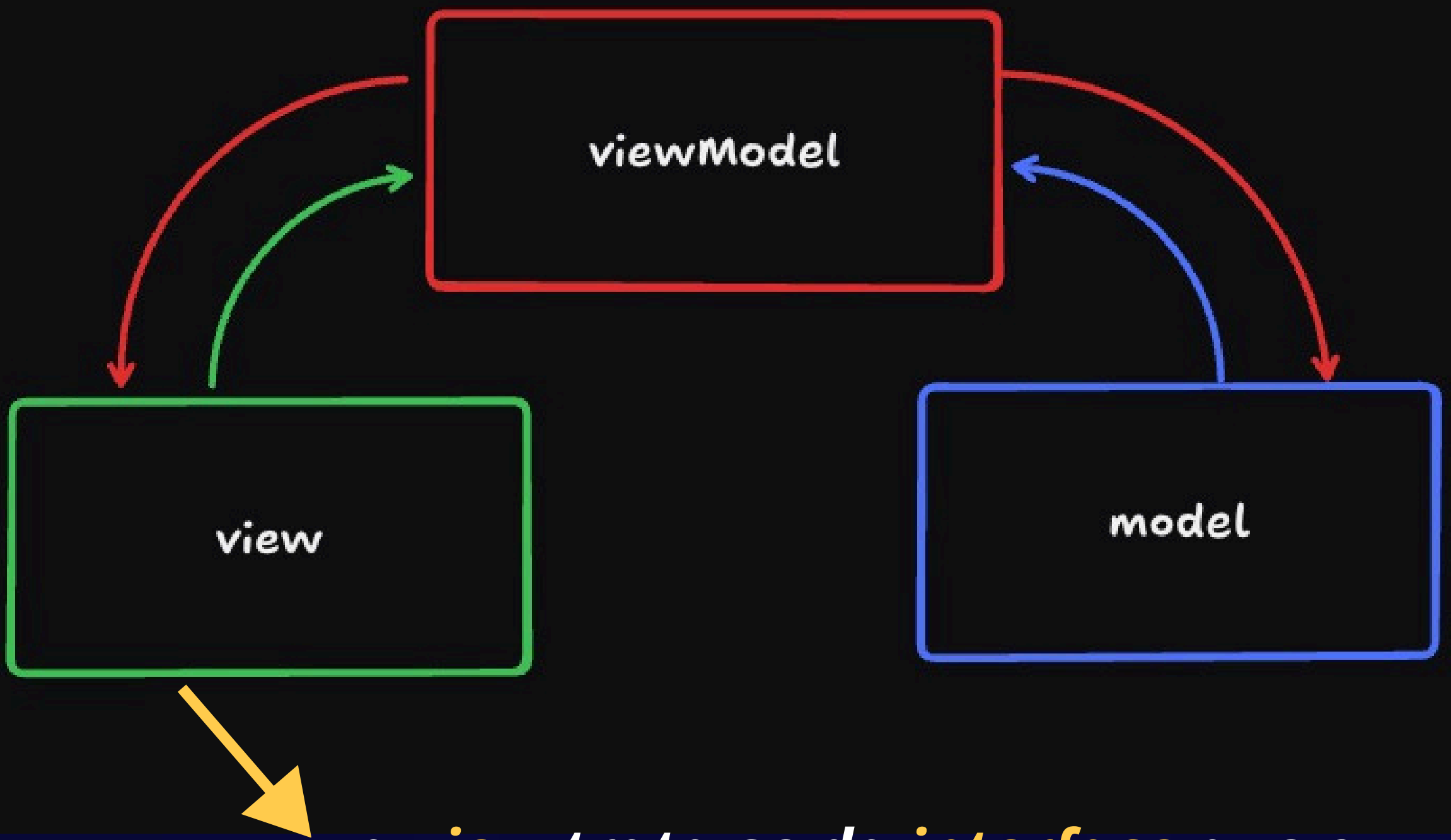
Desvendando arquitetura MVVM



E qual a diferença para custom hook Patterns ???

Isaac Gomes

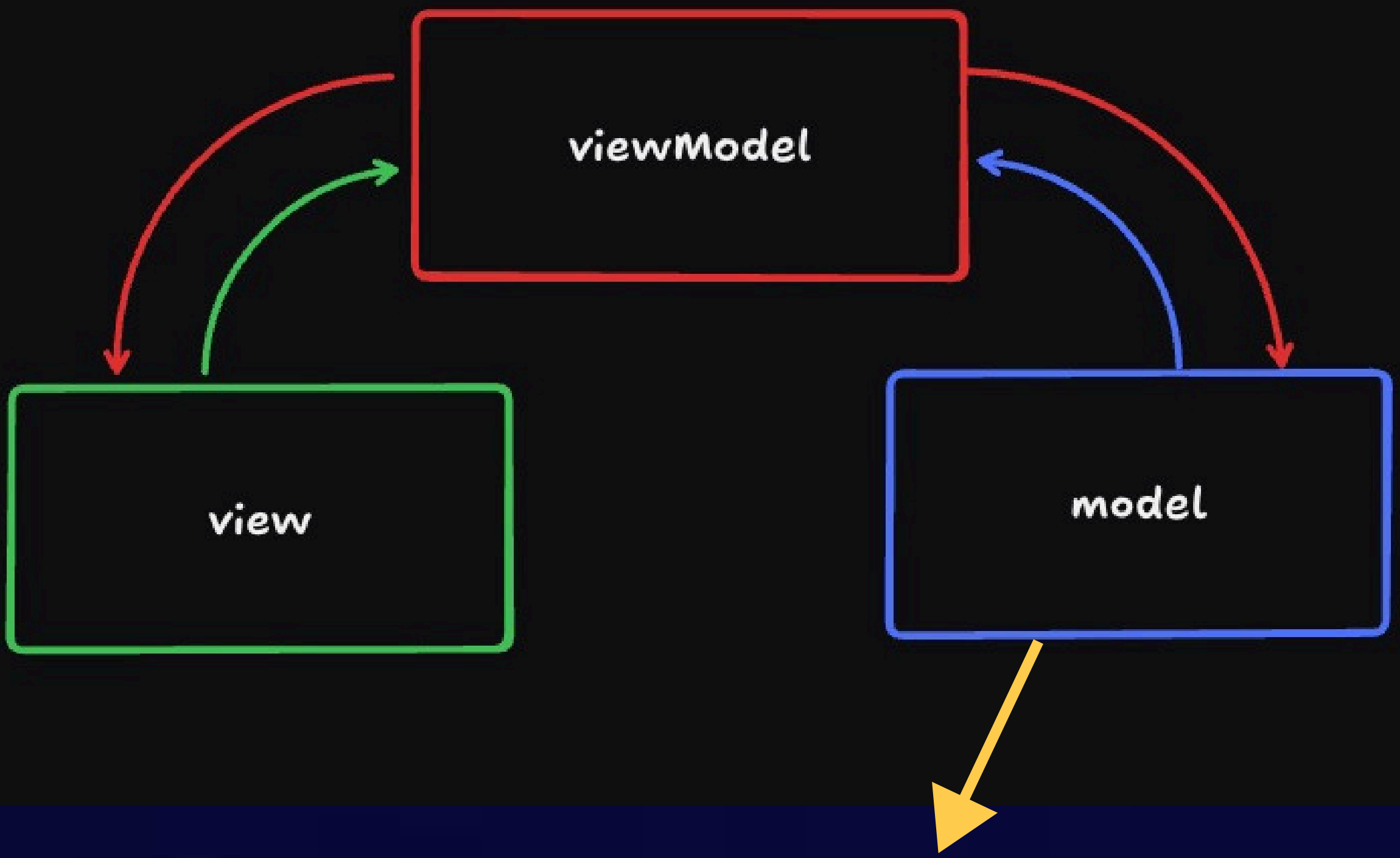
o que é MVVM ?



*a **view** trata-se da **interface** que o usuario sem logica. essa interface em termos gerais deve ser **burra** contendo a **menor** quantidade de **logica** possível*

Isaac Gomes

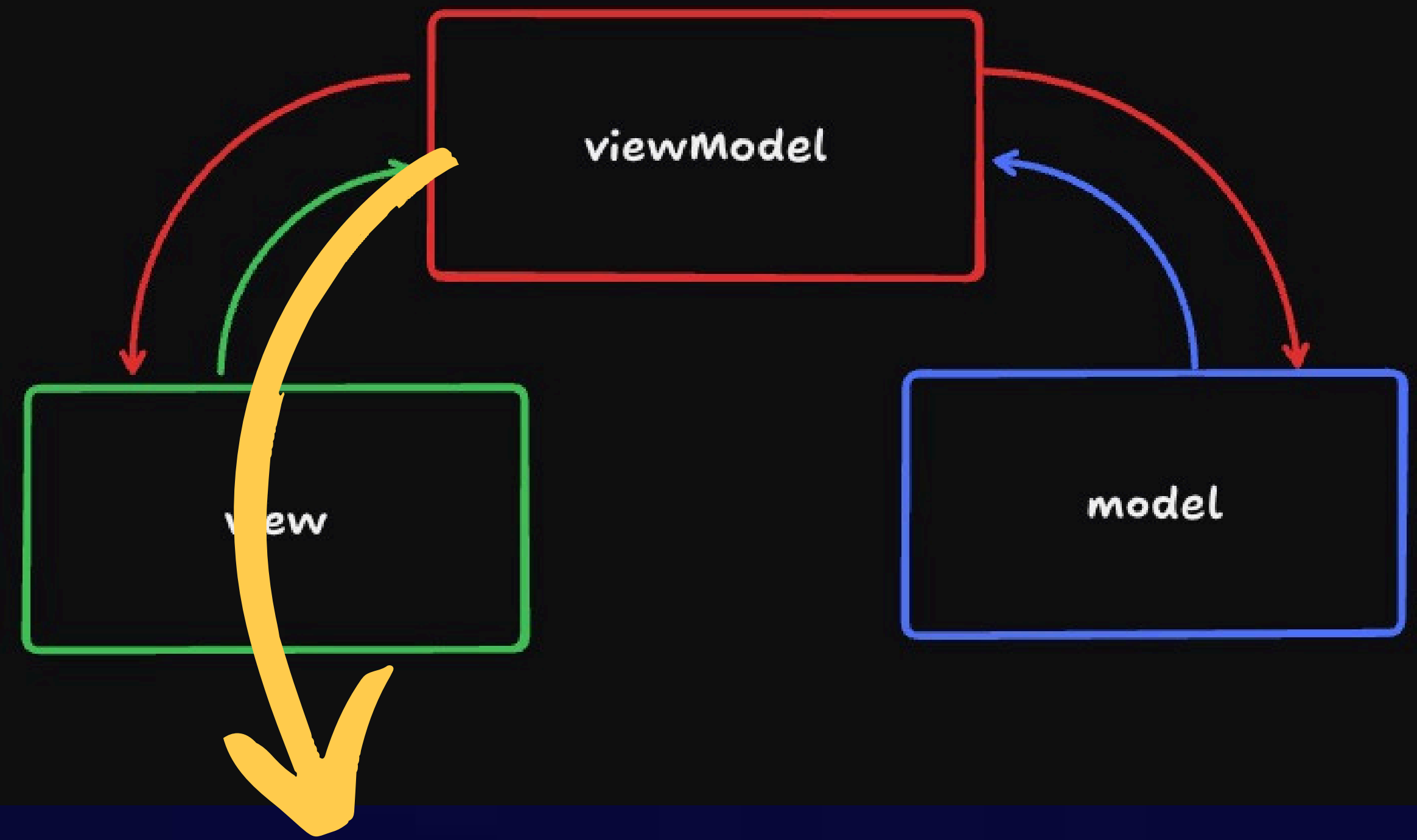
o que é MVVM ?



*o **Model** o model não contem nada visual somente **lógica de negócios** da aplicação*

Isaac Gomes

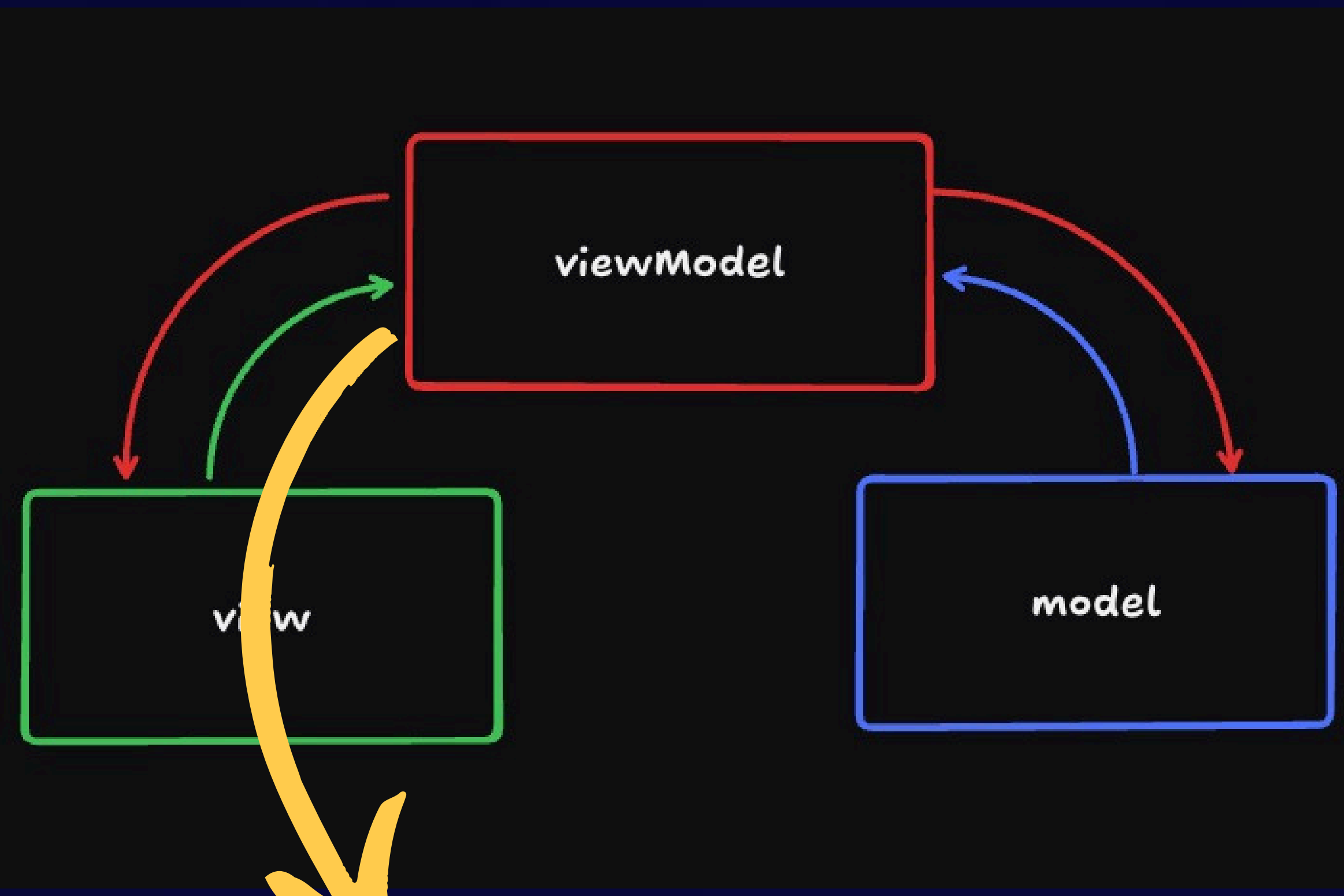
o que é MVVM ?



a viewModel Expõe os dados e os comandos que a View pode ligar (binding) ou seja ela Atua como um intermediário entre a View e o Model

Isaac Gomes

o que é MVVM ?

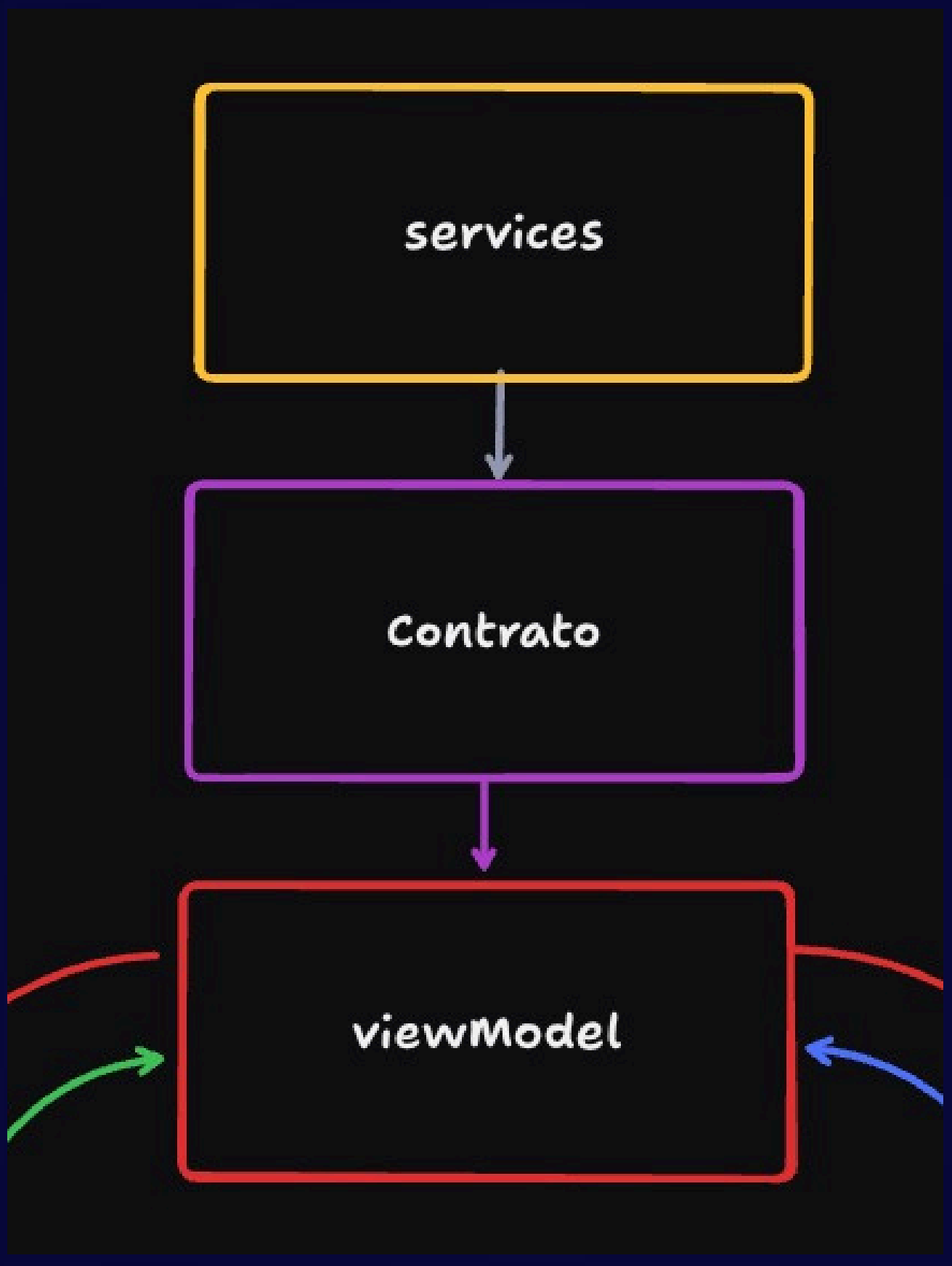


uma duvida muito comum

é qual a diferença entre
MVVM e Custom Hook Patterns??
afinal para que essa
viewModel no meio serve???

Isaac Gomes

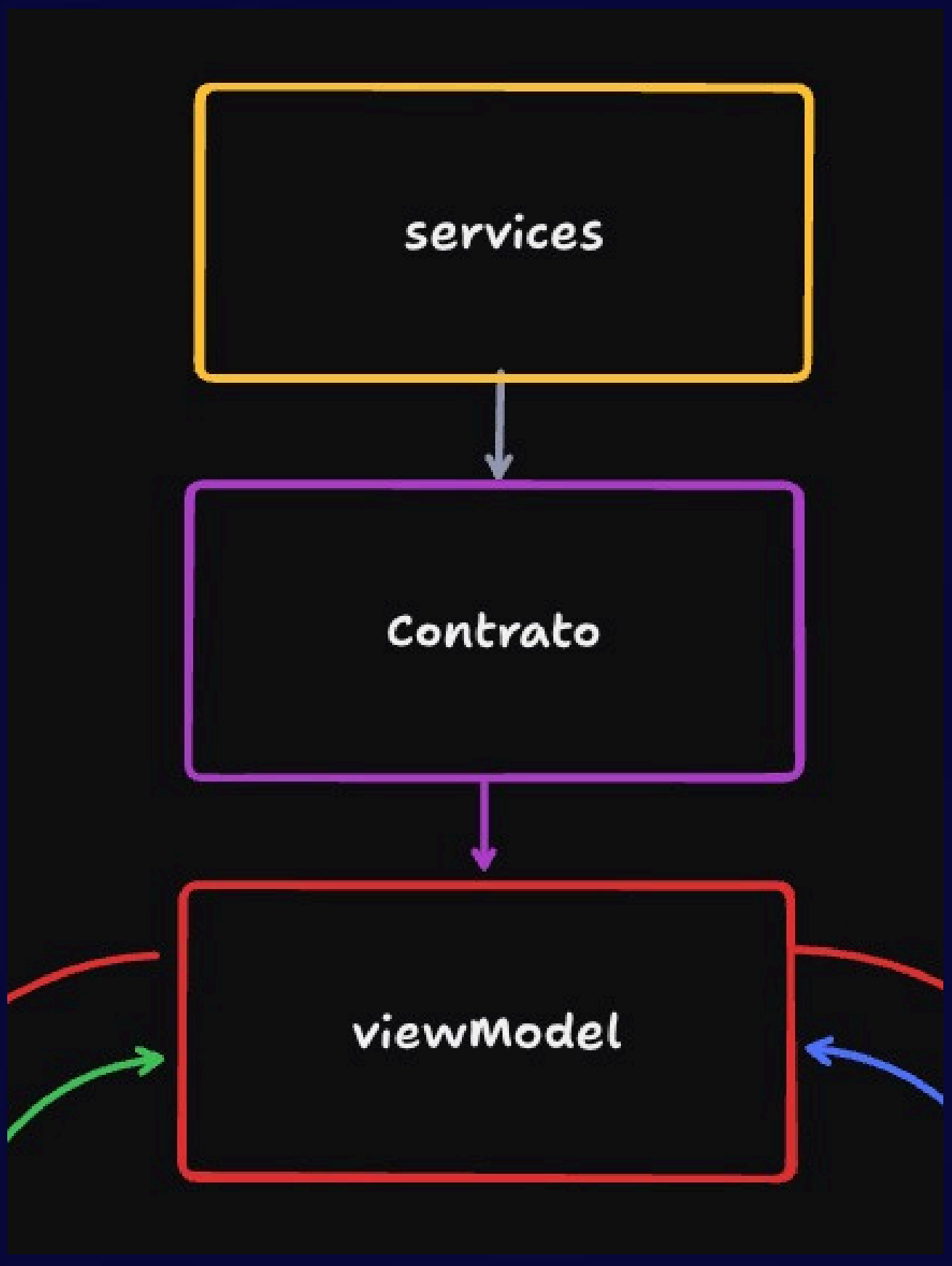
o que é MVVM ?



a **viewModel** alem de unir a **view** e o **model** pode ser um **orquestrado de recursos**, pense que vai acessar uma API podemos criar **componentes** muito claros do nosso **Software**

Isaac Gomes

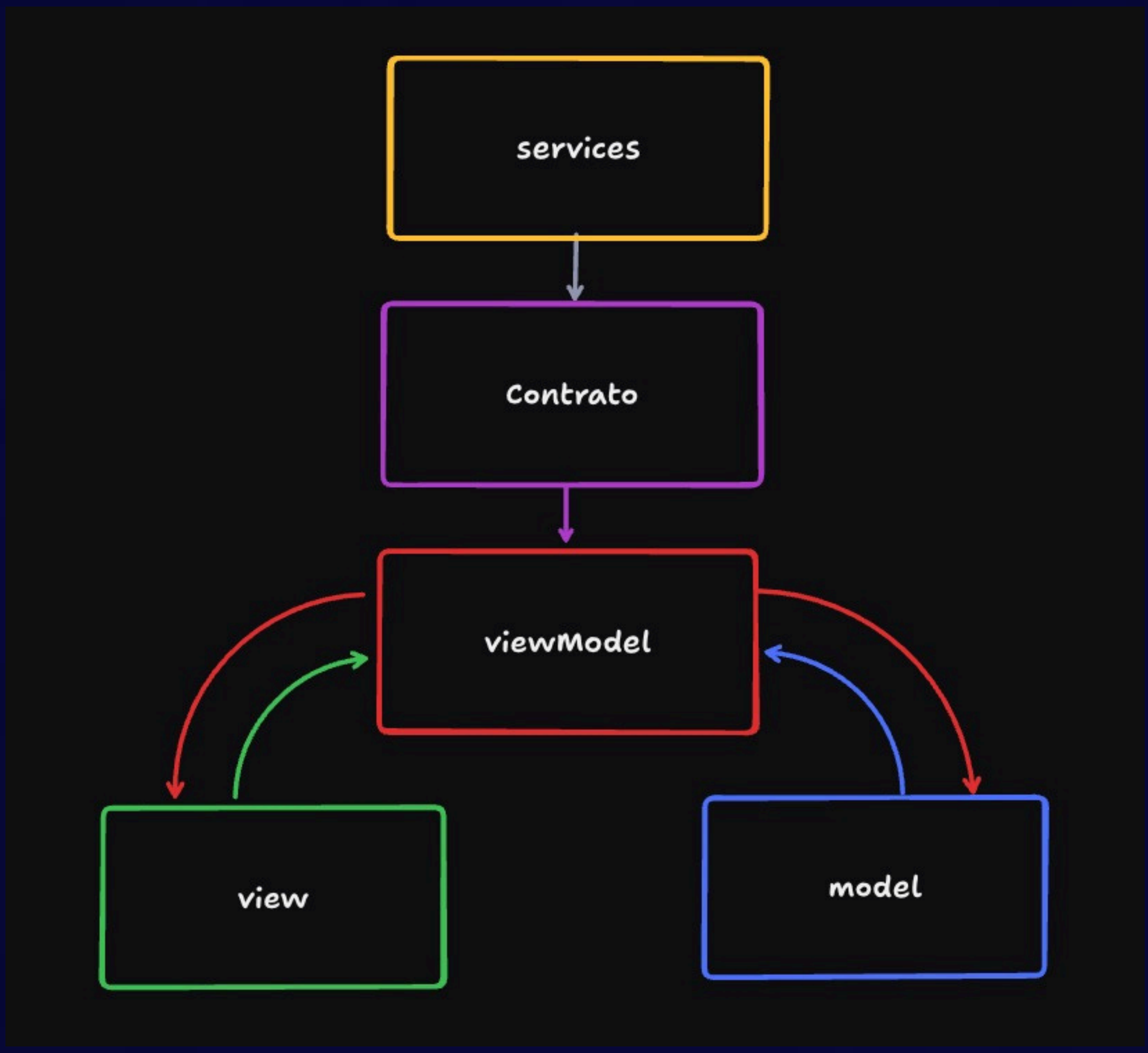
o que é MVVM ?



a **viewModel** alem de unir a **view** e o **model** pode ser um **orquestrado de recursos**, pense que vai acessar uma API podemos criar **componentes** muito claros do nosso **Software**

Isaac Gomes

o que é MVVM ?



Isaac Gomes

o que é MVVM ?

agora vamos montar um exemplo para ficar mais claro

cenário

vamos criar um todo list com as funções de listar, adicionar e deletar

Isaac Gomes

1 - criar uma interface para as funções de acesso a API

```ts
export interface ITaskService {
  create: (text: string) => Promise<void>
  remove: (id: string) => Promise<void>
  getAll: () => Promise<Array<Task>>
}
```

Isaac Gomes

2 - Criar o Service de acesso a API

```
export class TaskService implements ITaskService {
  async create(text: string) {
    const { data } = await apiCore.post('/task/', { text })
    return data
  }

  async remove(id: string) {
    const { data } = await apiCore.delete(`/task/${id}`)
    return data
  }

  async getAll() {
    const { data } = await apiCore.get('/task')
    return data
  }
}
```

Isaac Gomes

TS

3 - criar a logica do nosso TODO usando a interface definida de acesso a API

```
export function useTodo(service: ITaskService) {
  const [taskText, setTaskText] = useState<string>('')
  const queryClient = useQueryClient()

  const invalidateTaskList = () => {
    queryClient.invalidateQueries({ queryKey: [QUERY_KEY.TODO_LIST] })
  }

  const { data: ListTasks, isLoading: isLoadinglistTask } = useQueryListTask({
    service: service.getAll,
  })

  const { mutate: mutateAddTask } = useMutationAddTask({
    service: service.create,
    onSuccess: invalidateTaskList,
  })

  const { mutate: mutateDeleteTask } = useMutationRemoveTask({
    service: service.remove,
    onSuccess: invalidateTaskList,
  })

  const handleAddTask = () => {
    if (!taskText) return
    mutateAddTask(taskText)
    setTaskText('')
  }

  return {
    taskText, setTaskText, ListTasks,isLoadinglistTask,
    handleAddTask,mutateDeleteTask,
  }
```

Isaac Gomes

4 - criar a nossa interface/view

```tsx
export function TodoView(props: ReturnType<typeof useTodo>) {
  const { taskText, setTaskText, ListTasks, handleAddTask } = props
  return (
    <div>
      <div>
        <input
          type="text"
          value={taskText}
          onChange={({ target: { value } }) => setTaskText(value)}
        />
        <button onClick={handleAddTask}> adiconar </button>
      </div>

      <div>{ListTasks?.length}</div>
      <div>
        {ListTasks?.map((task) => (
          <div key={task.id}>
            <p>{task.text}</p>
            <button>deletar</button>
          </div>
        ))}
      </div>
    </div>
  )
}
```

Isaac Gomes

5 - criar nossa viewModel orquestrando nossos Componentes desenhados

```tsx
export default function Todo() {
  const taskService = new TaskService()
  const methods = useTodo(taskService)

  return <TodoView {...methods} />
}
```

Isaac Gomes

6 - agora vamos criar uma implementação falsa do acesso a API para fazer os testes

```
let tasks: Array<Task> = []
export function useTaskServiceMock(): ITaskService {
  const create = async (text: string): Promise<void> => {
    const task: Task = {
      id: String(Math.random()),
      text,
    }
    tasks = [...tasks, task]
  }

  const remove = async (id: string): Promise<void> => {
    tasks = tasks.filter((item) => item.id !== id)
  }

  const getAll = async (): Promise<Task[]> => {
    return tasks
  }

  return { create, remove, getAll }
}
```

aqui começa a ficar interessante vemos que estamos mockando um acesso a API e criando o comportamento esperado

Isaac Gomes

o **MVVM** é sempre mencionada pela **testabilidade**, agora vamos ver isso na pratica

quero testar meu Model

```
export const ModelTest = () => {
  const taskServiceMock = useTaskServiceMock()
  const methods = useTodo(taskServiceMock)
  return methods
}
```

Pronto podemos testar nossa logica

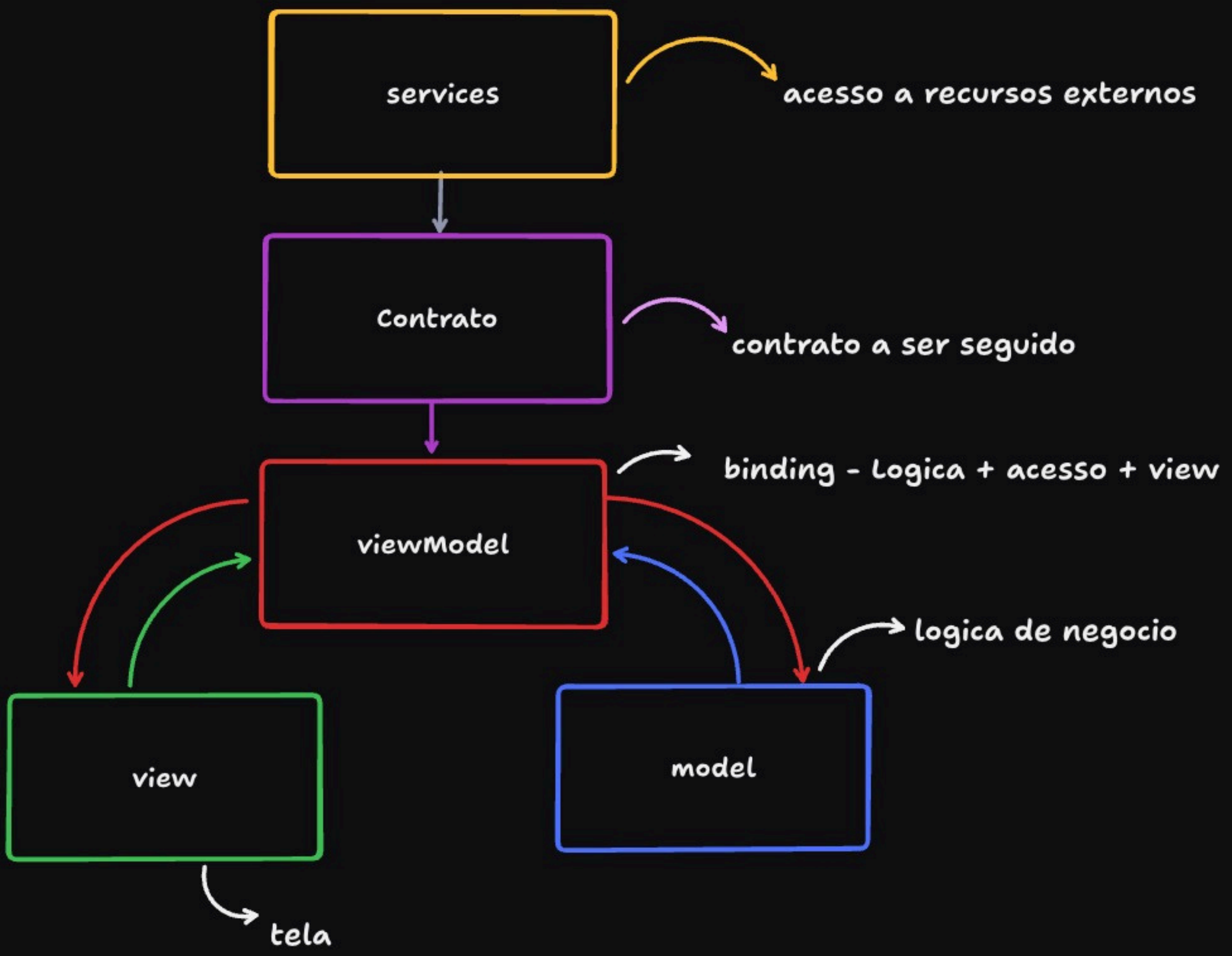
quero testar minha viewModel

```
export const ViewModelTest = ()=> {
  const taskServiceMock = useTaskServiceMock()
  const methods = useTodo(taskServiceMock)
  return <TodoView {...methods} />
}
```

pronto podemos testar nossa viewModel e conferir se a união esta como esperado

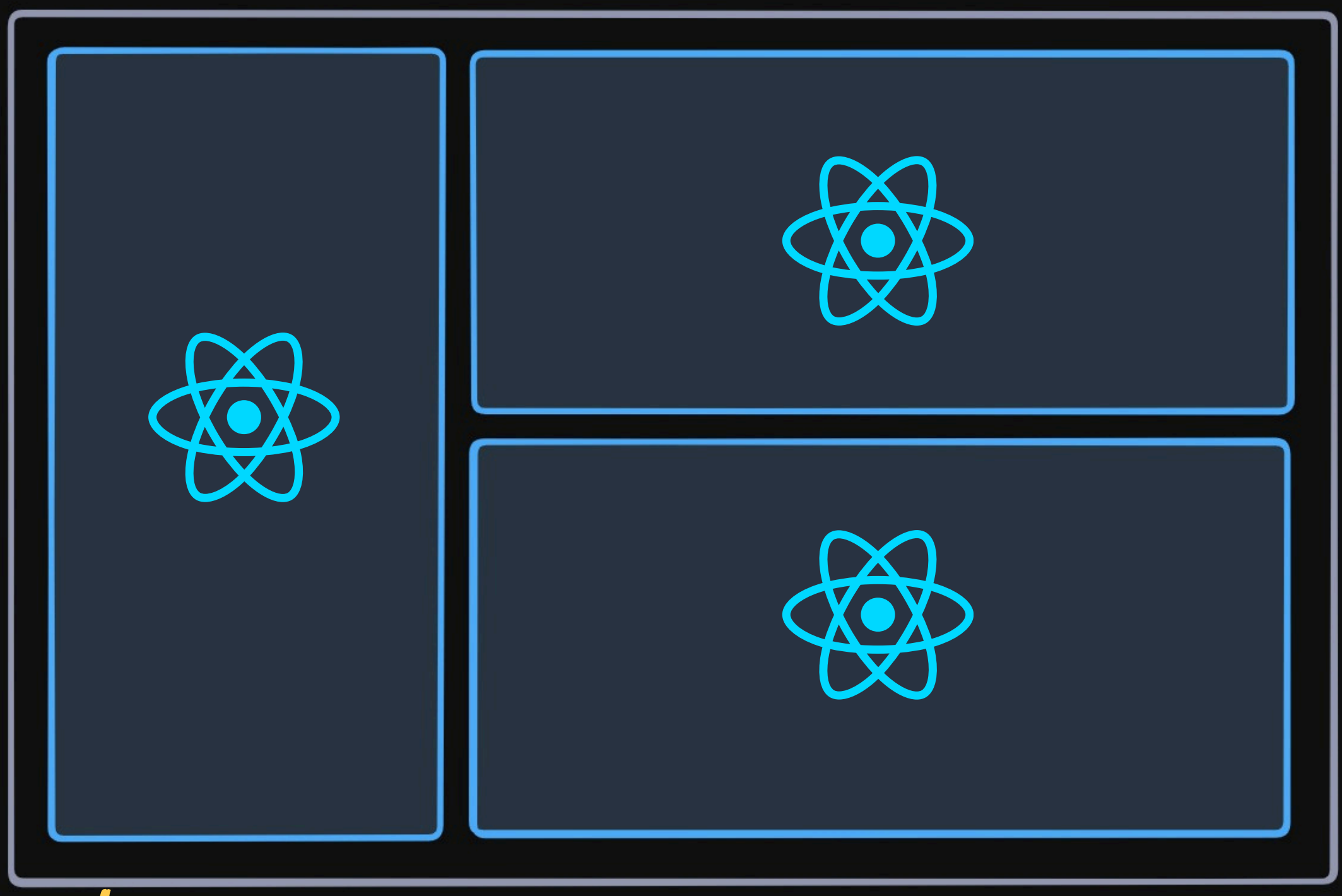
Isaac Gomes

componentes da arquitetura



Isaac Gomes

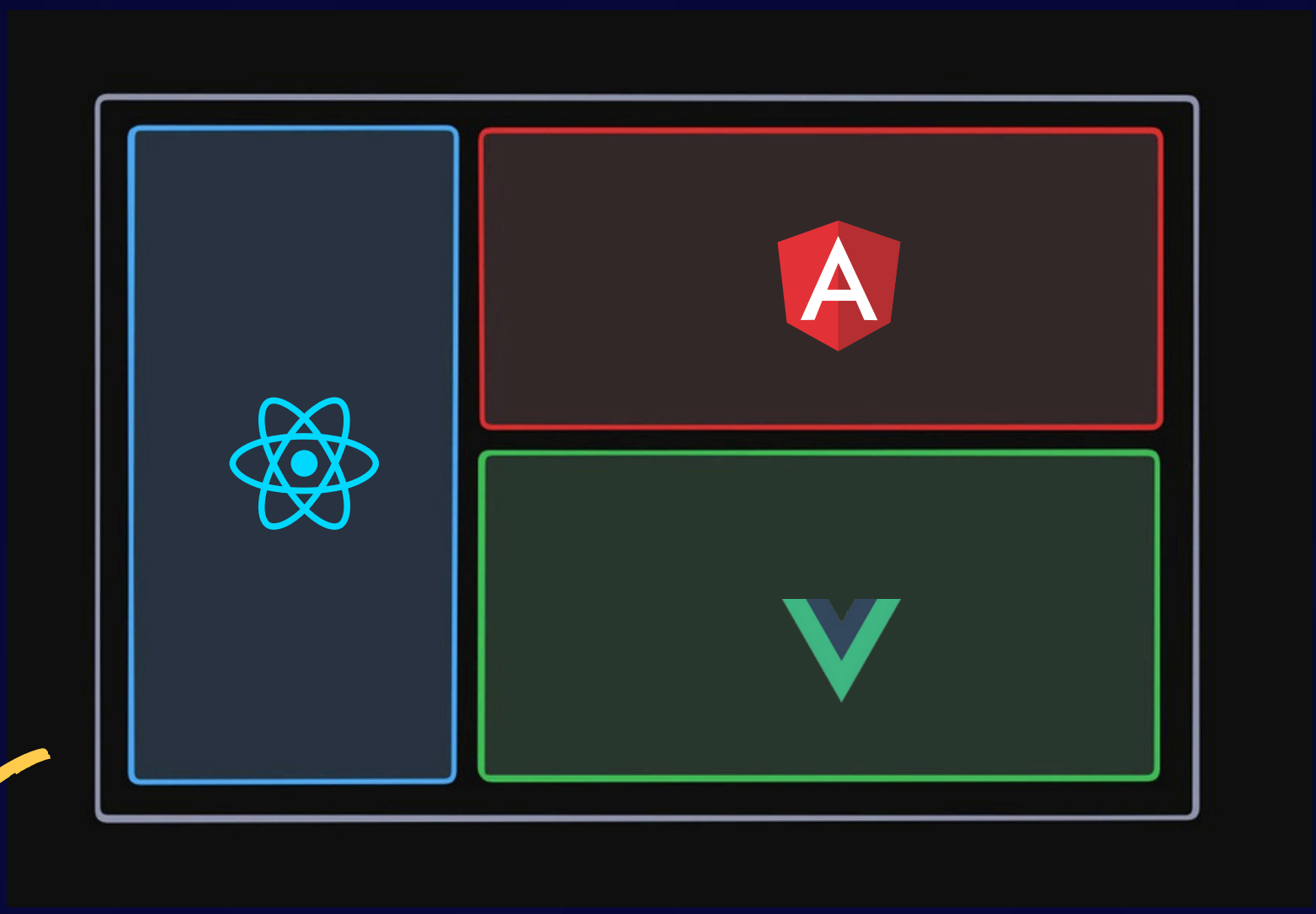
Desvendando Arquitetura de MICRO-FRONTENDS



Exemplo com React.js

Isaac Gomes

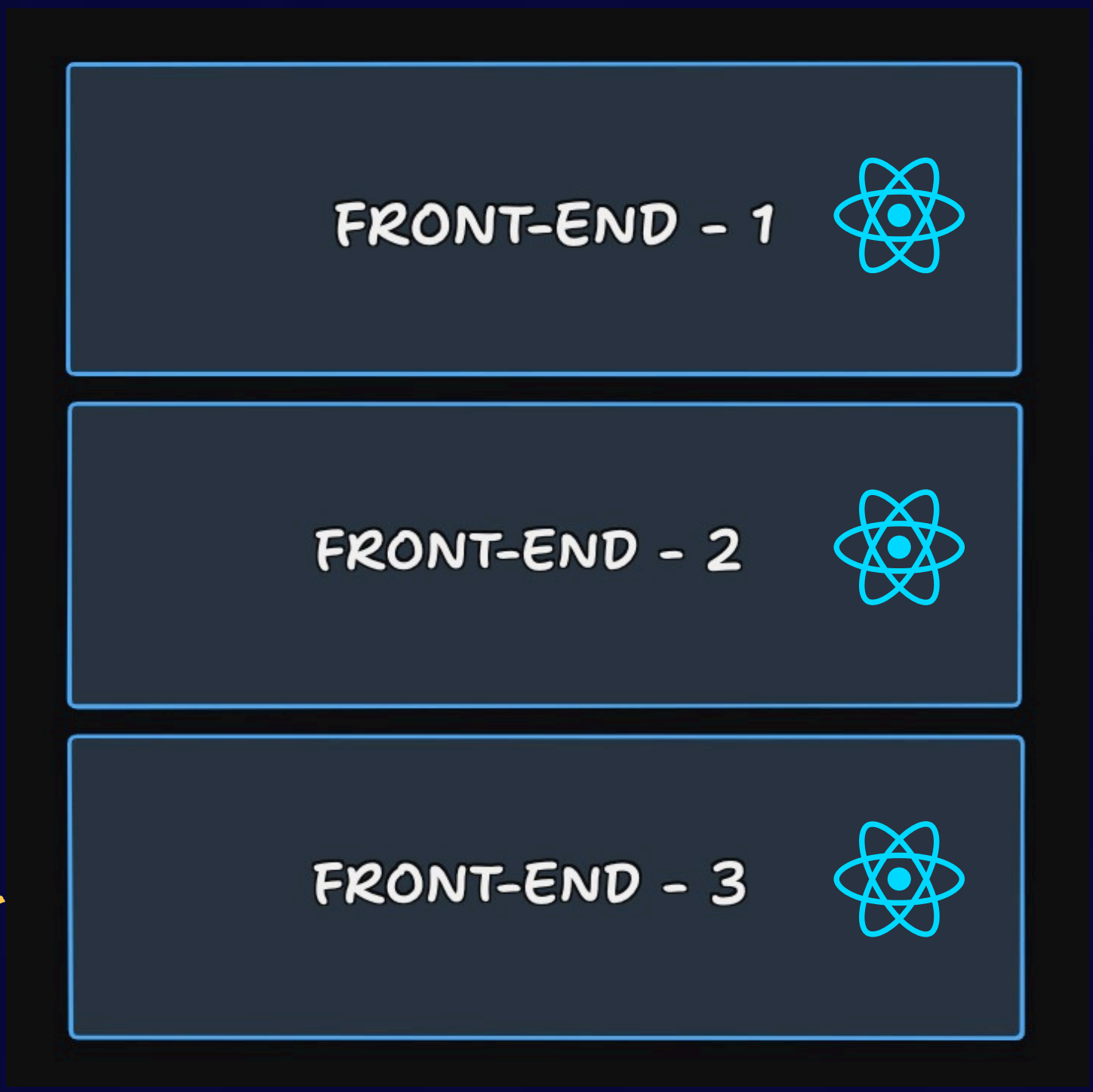
o que é MICRO-FRONTENDS ?



***Micro-frontends** dividem uma aplicação front-end em **partes menores**, **autônomas** e **independentes**, desenvolvidas e mantidas por equipes diferentes, permitindo o uso de **tecnologias variadas**.*

Isaac Gomes

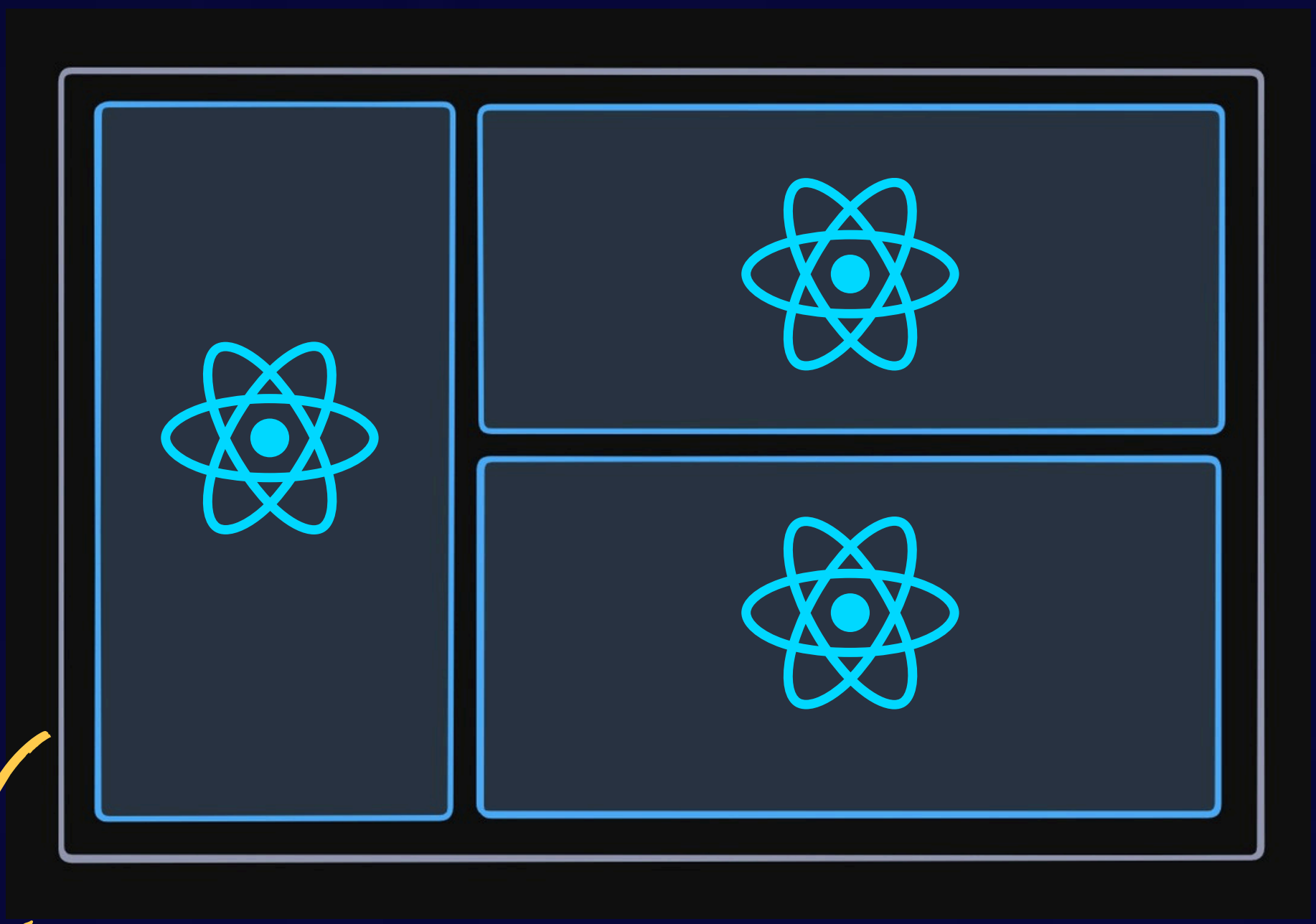
o que é MICRO-FRONTENDS ?



Cada micro-frontends funciona como um módulo separado, integrado na aplicação final, promovendo escalabilidade e flexibilidade, mas exigindo cuidado com a integração e performance.

Isaac Gomes

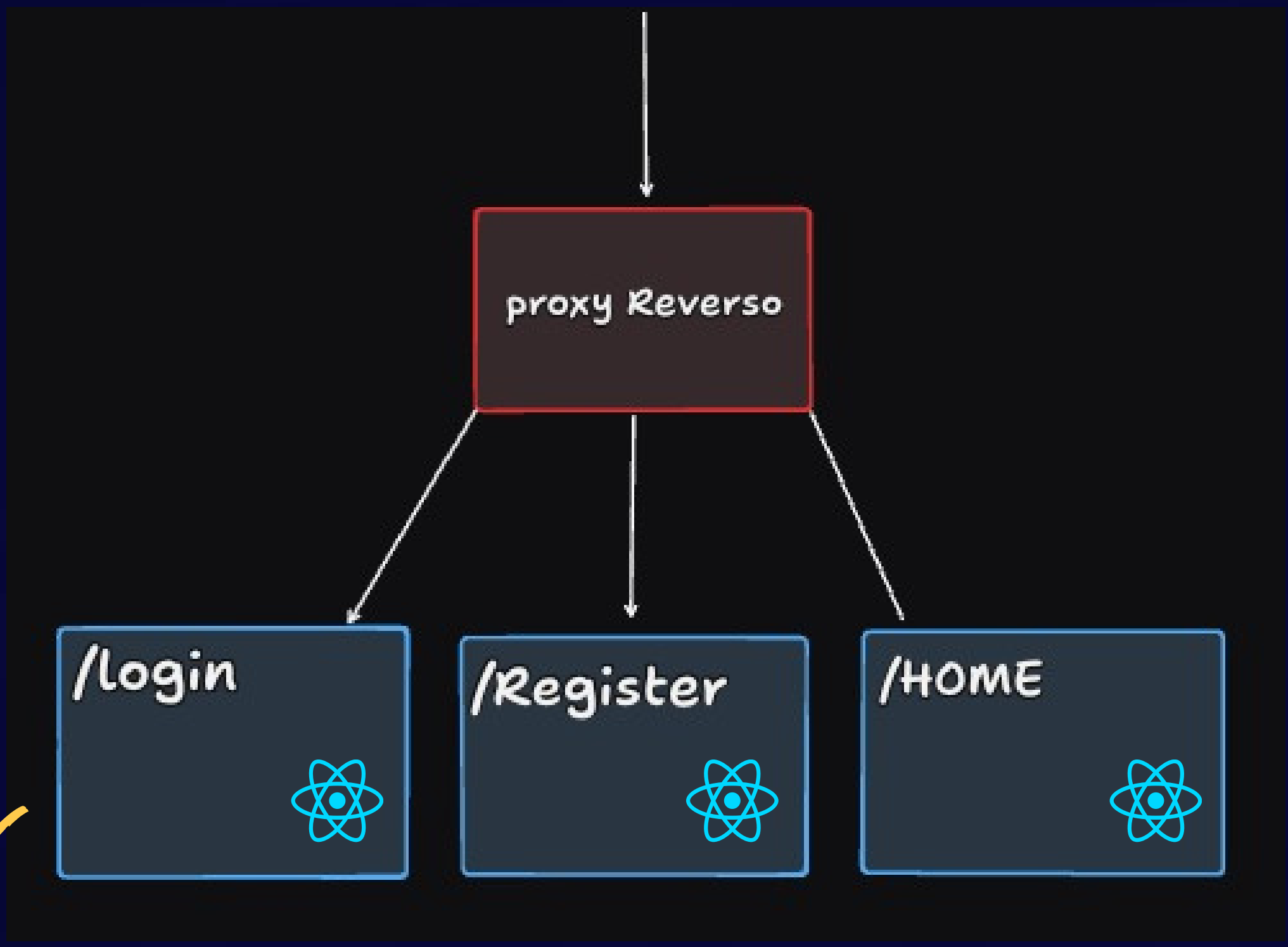
Tipos de MICRO-FRONTENDS



Parcel-Based MFE é uma abordagem de micro-frontends onde cada parte da aplicação é representada como um "parcel" ou módulo que pode ser carregado e renderizado de forma independente em qualquer ponto da aplicação

Isaac Gomes

Tipos de MICRO-FRONTENDS



Route-Based MFE é uma abordagem de micro-frontends onde cada micro-frontend é associado a uma rota específica dentro da aplicação

Isaac Gomes

como funciona MICRO-FRONTENDS com outras arquiteturas que vimos aqui?

A nivel de recurso podemos aplicar MVVM criando uma separação de (view, model e viewModel)

Isaac Gomes

TS

como funciona MICRO-FRONTENDS com outras arquiteturas que vimos aqui?

Cada MFE pode ter seu BFF

Isaac Gomes

TS

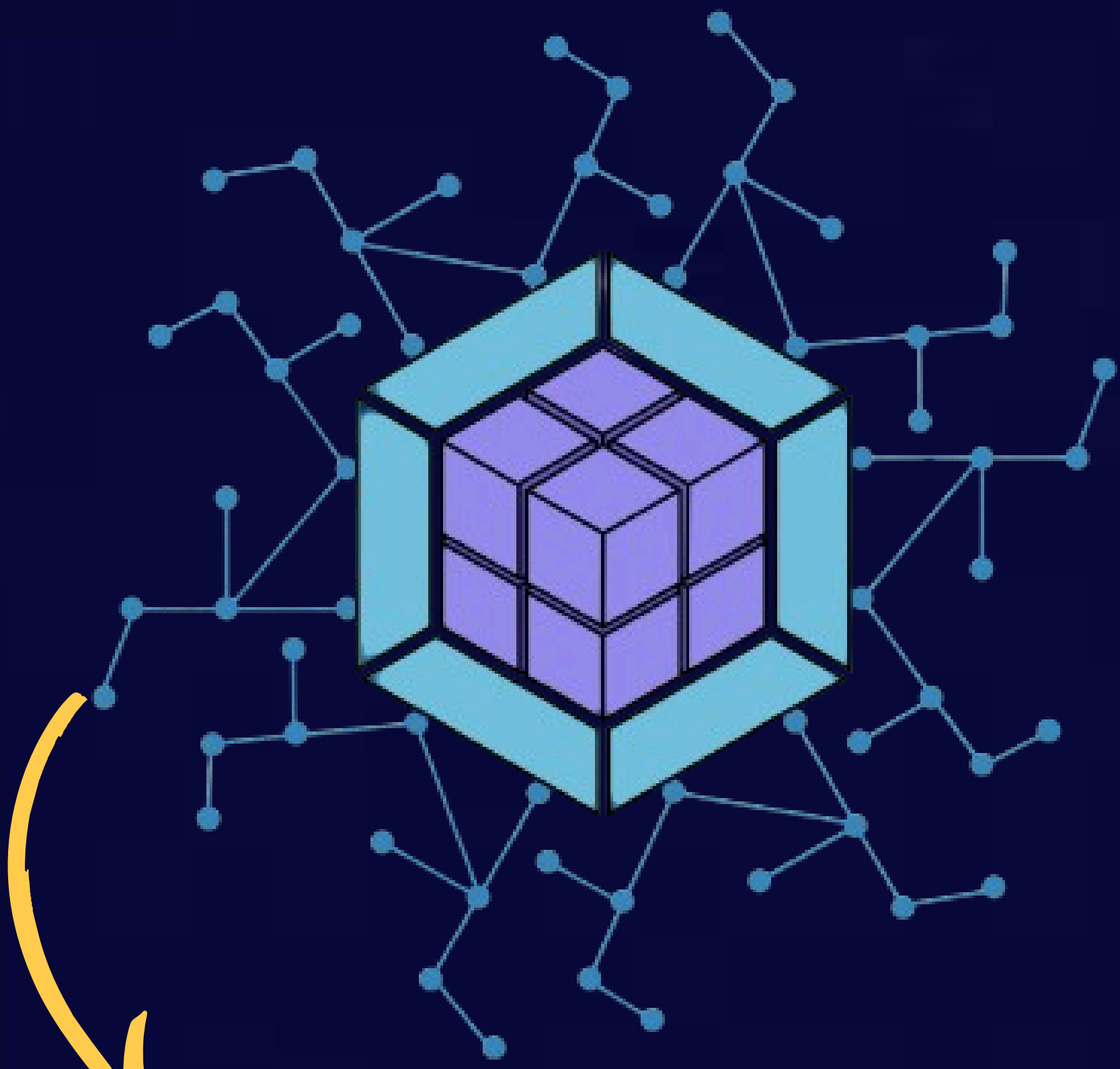
como funciona MICRO-FRONTENDS com outras arquiteturas que vimos aqui?

esses servicos poderiam usar
clean arch ou ports and adapter

Isaac Gomes

TS

como criar MICRO-FRONTENDS?



para exemplo usaremos webpack (Module Federation)

Isaac Gomes

como criar MICRO-FRONTENDS?

CONTAINER

homeAPP

ContactApp

para exemplos vamos criar 3 APPs
uma home, contact e um Container

Isaac Gomes

como criar MICRO-FRONTENDS?

```
new ModuleFederationPlugin({
      name: 'ContactApp',
      filename: 'remoteEntry.js',
      exposes: {
        './ContactApp': './src/Contact
      },
    }),
```

criamos um projeto com webpack e agora vamos configurar o ModuleFederation dizendo qual vai ser o nome e o que vamos expor

Isaac Gomes

TS

como criar MICRO-FRONTENDS?

```tsx
const Contact = () => {
  return (
      <Card>
        <img alt="Profile" src="https://picsum.photos/300/200" />
        <CardBody>
          <CardTitle tag="h5">John Doe</CardTitle>
          <CardSubtitle className="mb-2 text-muted" tag="h6">
            Software Engineer
          </CardSubtitle>
          <CardText>
            Email: john.doe@example.com
            <br />
            Phone: (123) 456-7890
          </CardText>
          <Button color="primary">Contact</Button>
        </CardBody>
    </Card>
  );
}
```

Aqui esta o nosso componente de Contact

Isaac Gomes

TS

como criar MICRO-FRONTENDS?

```
new ModuleFederationPlugin({
    name:  'HomeApp',
    filename: 'remoteEntry.js',
    exposes: {
       './HomeApp': './src/Home.
    },
})
```

agora vamos configurar a nossa HomeApp

Isaac Gomes

como criar MICRO-FRONTENDS?

```
const Home = () => {
  return (
    <Nav tabs>
      <NavItem>
        <NavLink href="#" active>
          Link
        </NavLink>
      </NavItem>
      <NavItem>
        <NavLink href="#">Link</NavLink>
      </NavItem>
      <NavItem>
        <NavLink href="#">Another Link</NavLink>
      </NavItem>
      <NavItem>
        <NavLink disabled href="#">
          Disabled Link
        </NavLink>
      </NavItem>
    </Nav>
  )
}
```

Aqui esta o nosso componente de Home

Isaac Gomes

TS

como criar MICRO-FRONTENDS?

```
new ModuleFederationPlugin({
    name: 'App',
    remotes: {
      HomeApp: 'HomeApp@http://localhost:9002/remoteEntry.js',
      ContactApp: 'ContactApp@http://localhost:9003/remoteEntry.js',
    },
}),
```

agora vamos criar nosso container e no ModuleFederation dele vamos declarar o que vão acessar "remotes"

Isaac Gomes

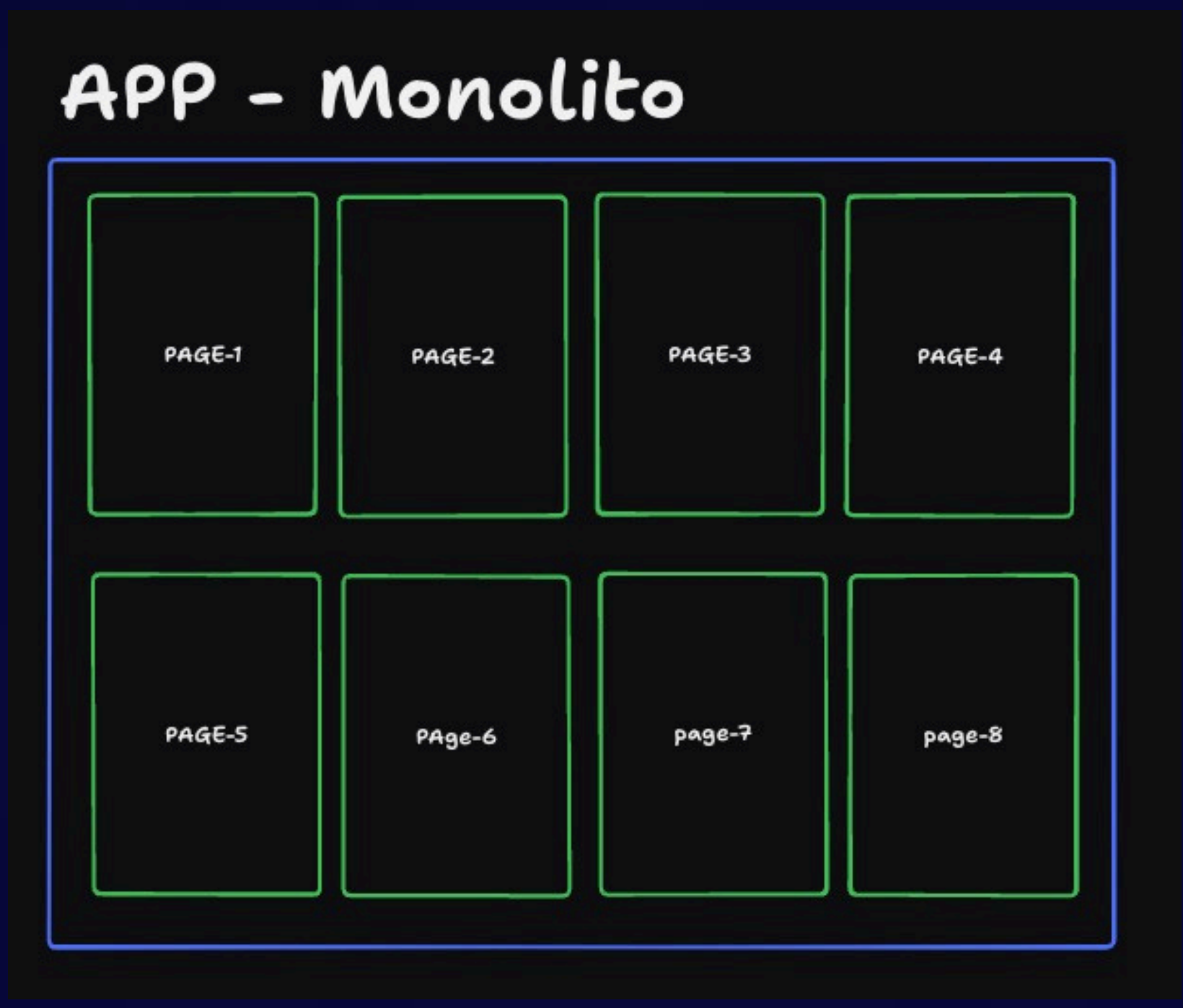
como criar MICRO-FRONTENDS?

```tsx
import HomePage from 'HomeApp/HomeApp'
import ContactApp from 'ContactApp/ContactApp'

const App = () => {
  return (
    <div>
      <p>CONTAINER</p>
      <HomePage />
      <ContactApp />
    </div>
  )
}
```

agora é só usar os MF criados

Isaac Gomes

MONOLITO / MICRO-FRONTENDS

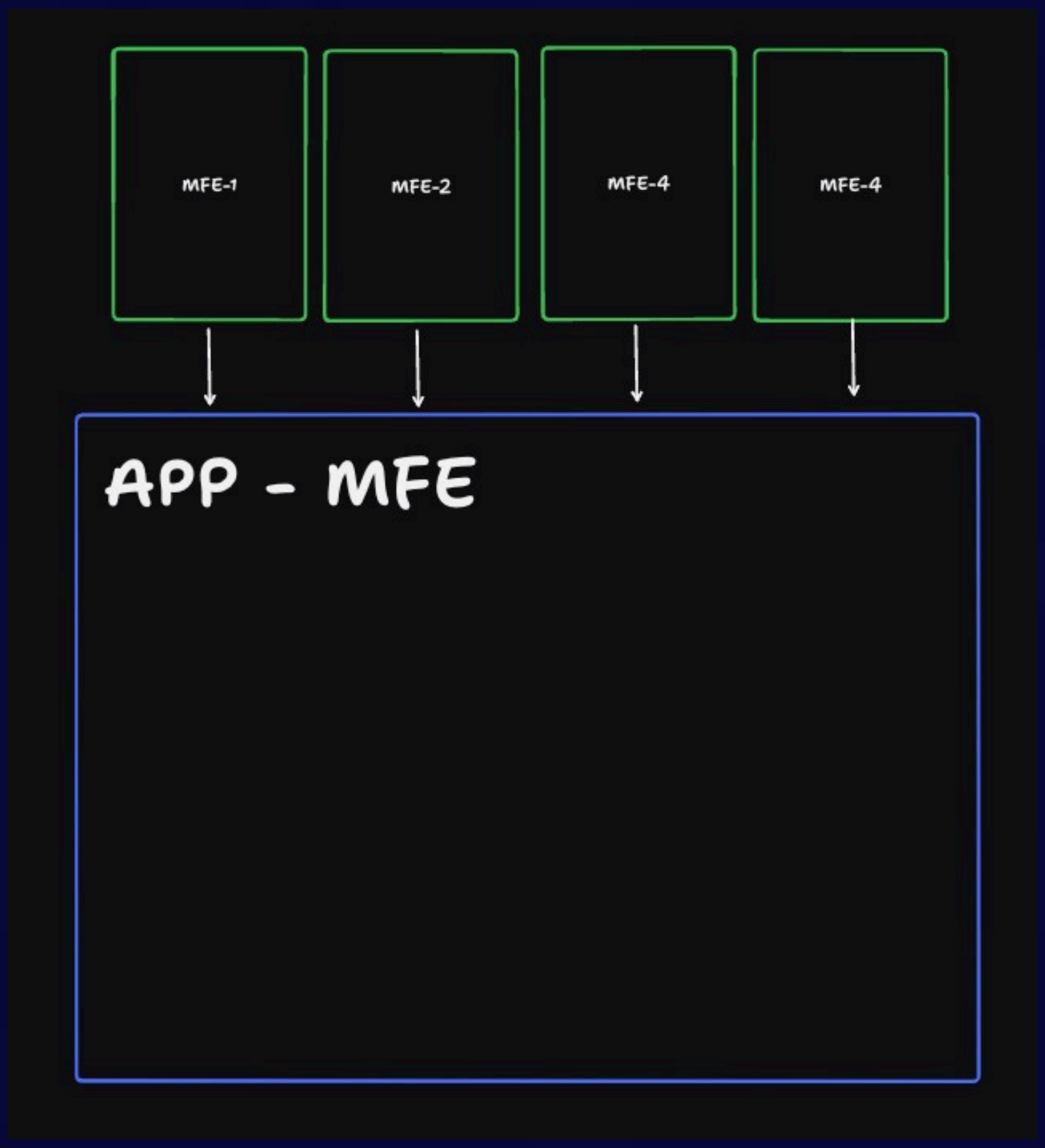


então aqui podemos ver a diferença de quando temos um unico monolito frontend que tem uma Codebase única

Isaac Gomes

TS

MONOLITO / MICRO-FRONTENDS



e um projeto com **MFE** com essa divisão da codebase mais **modular** e **independente**

Isaac Gomes

Gostou?

Curta

Compartilhe

Salve

Isaac Gomes