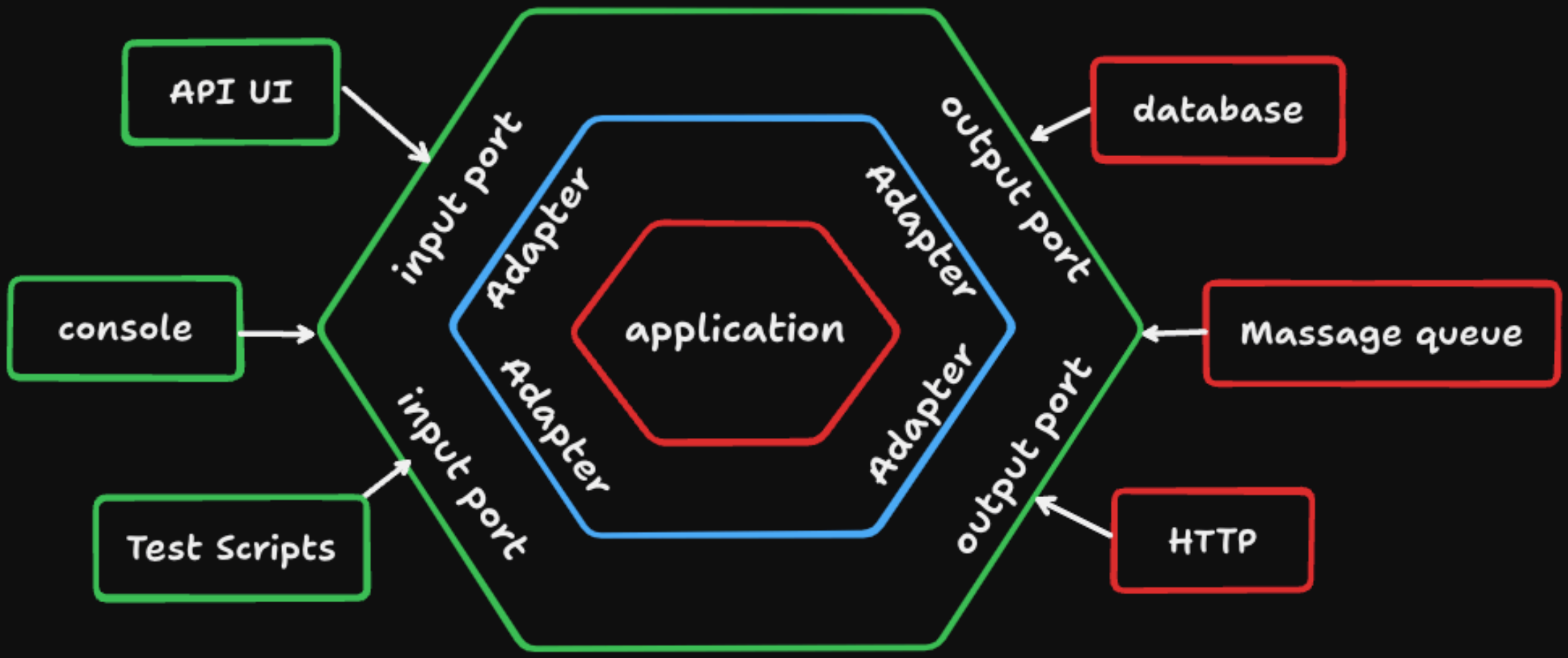
Desvendando arquitetura hexagonal

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?

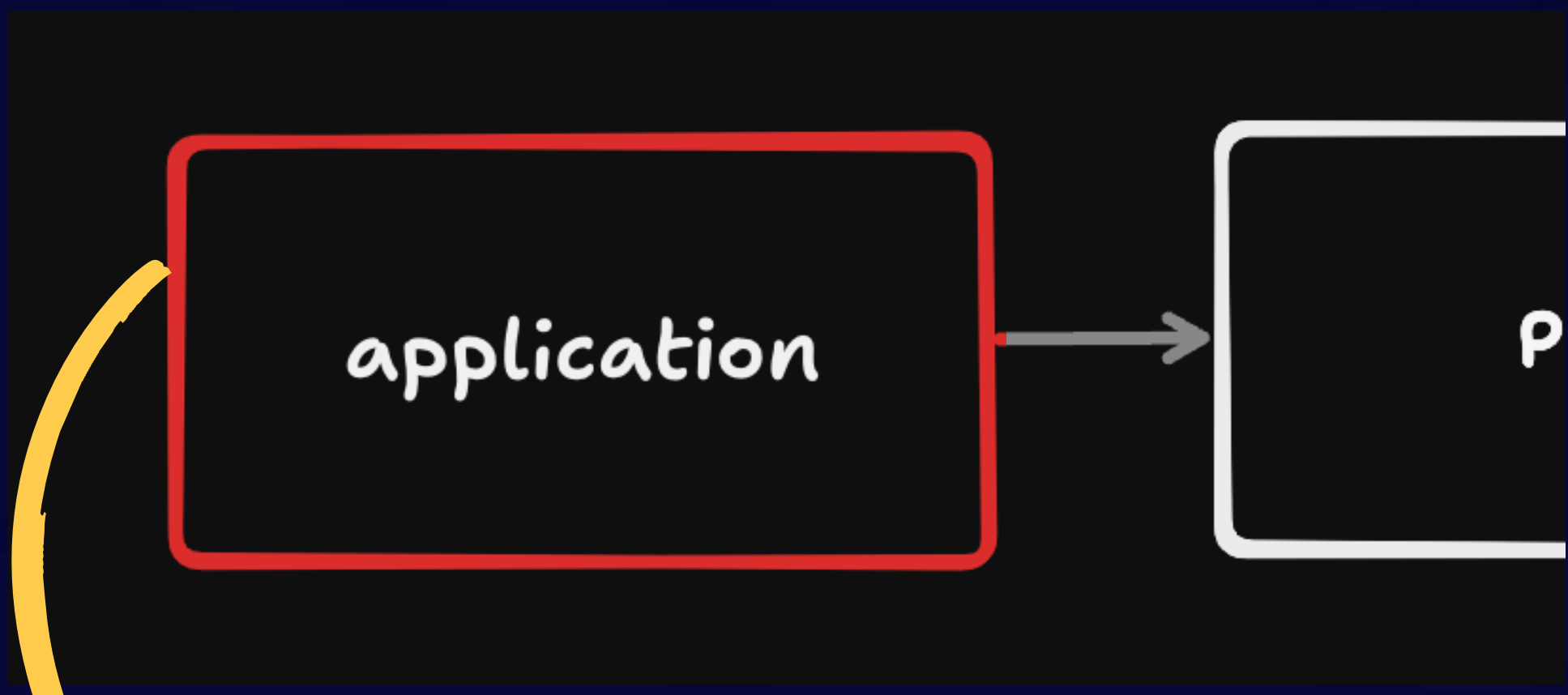
Isaac Gomes

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

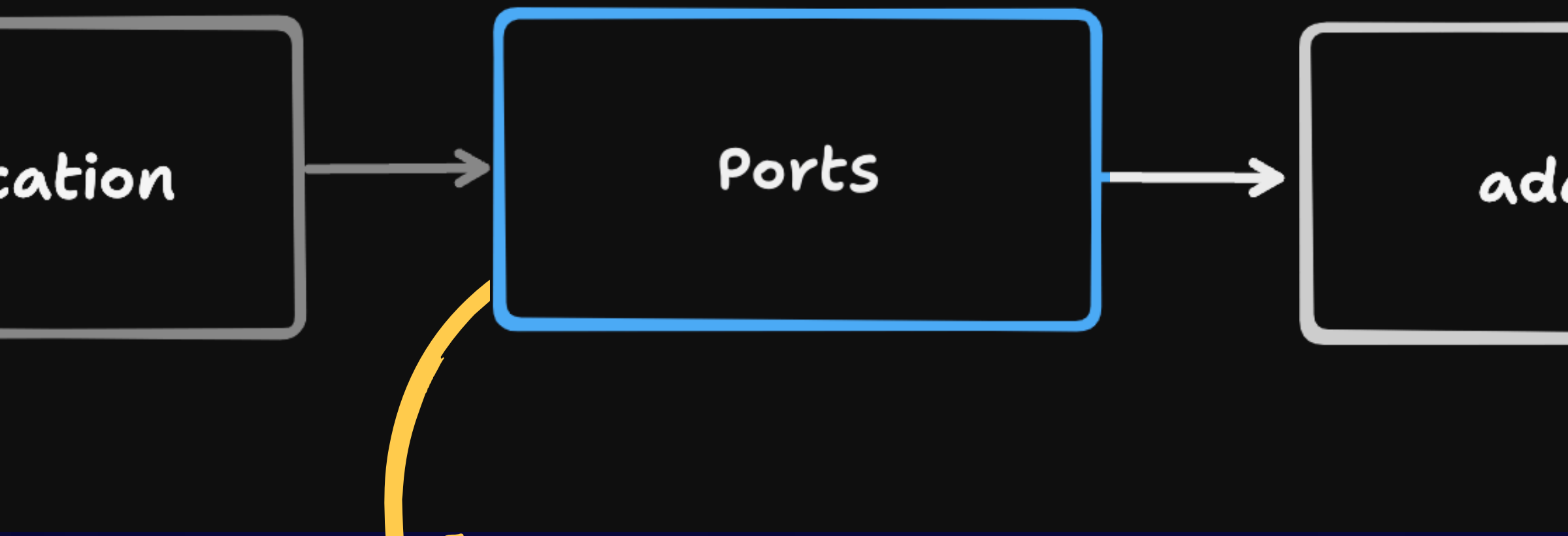
arquitetura hexagonal



Contém os casos de uso que orquestram a lógica de negócios

Isaac Gomes

arquitetura hexagonal

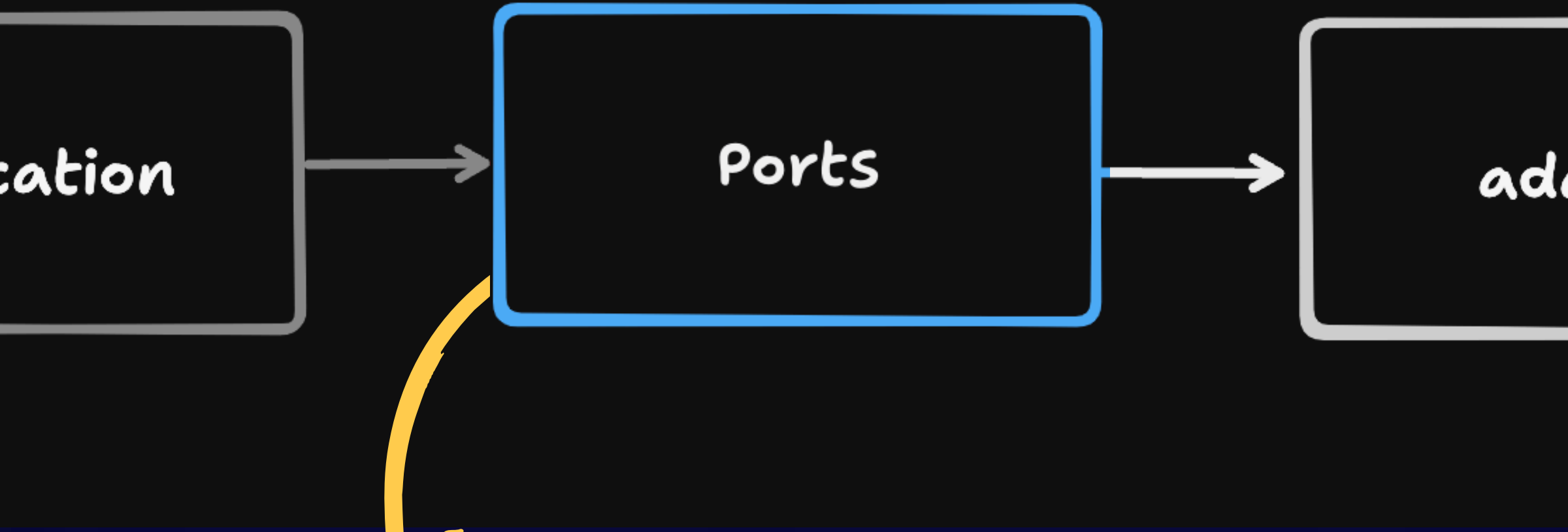


Inbound Ports

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

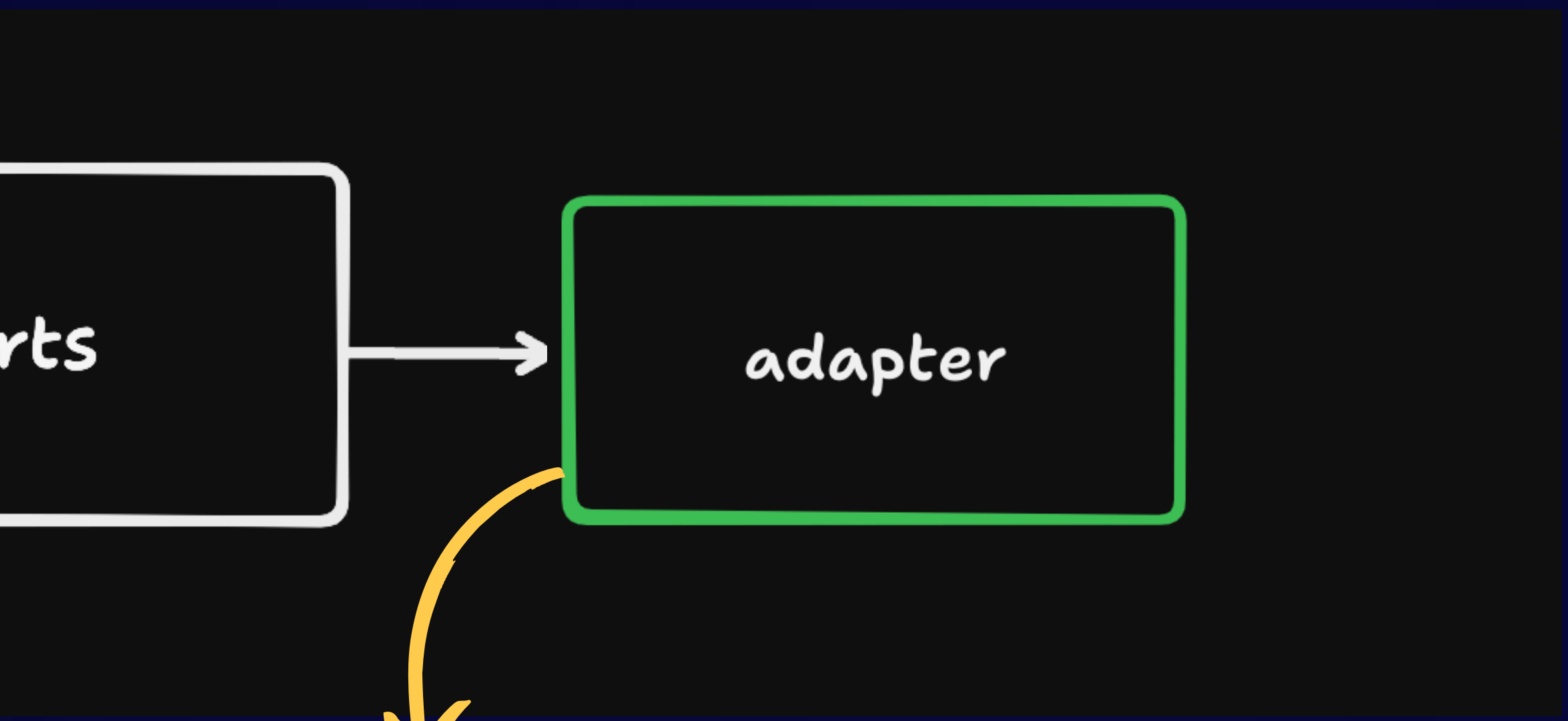


Outbound Ports

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

arquitetura hexagonal

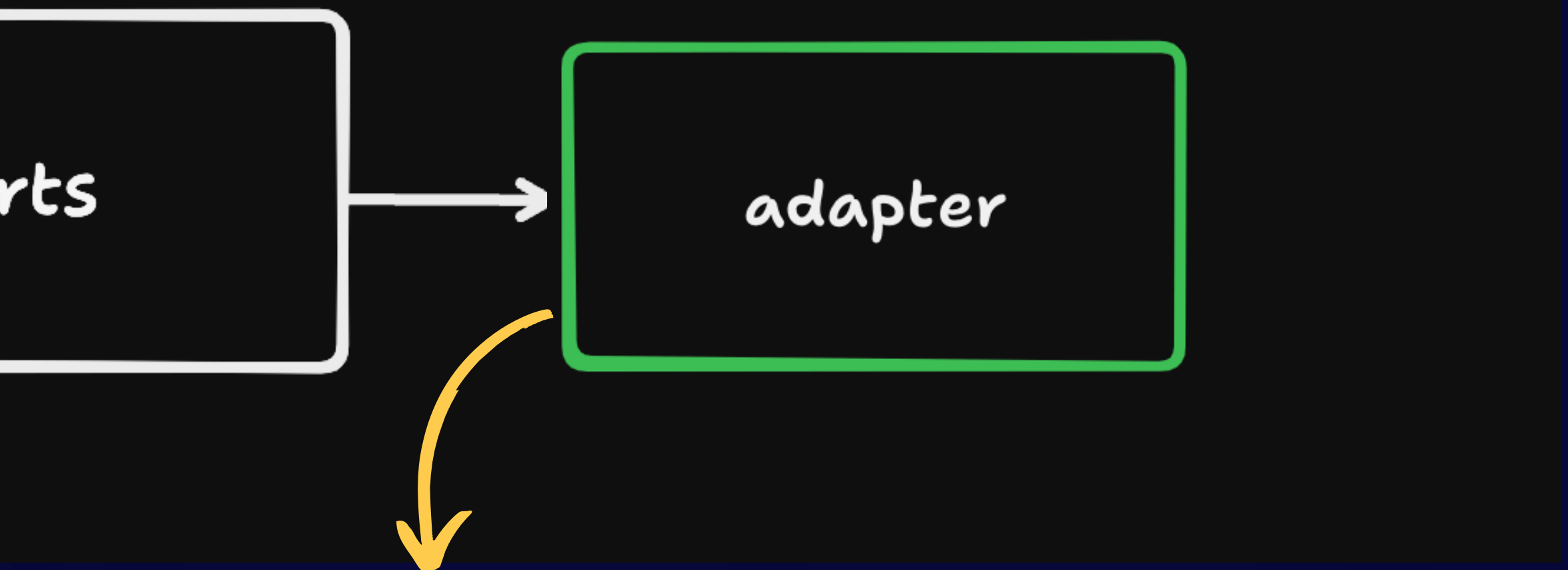


Inbound Adapters

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

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

TS

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

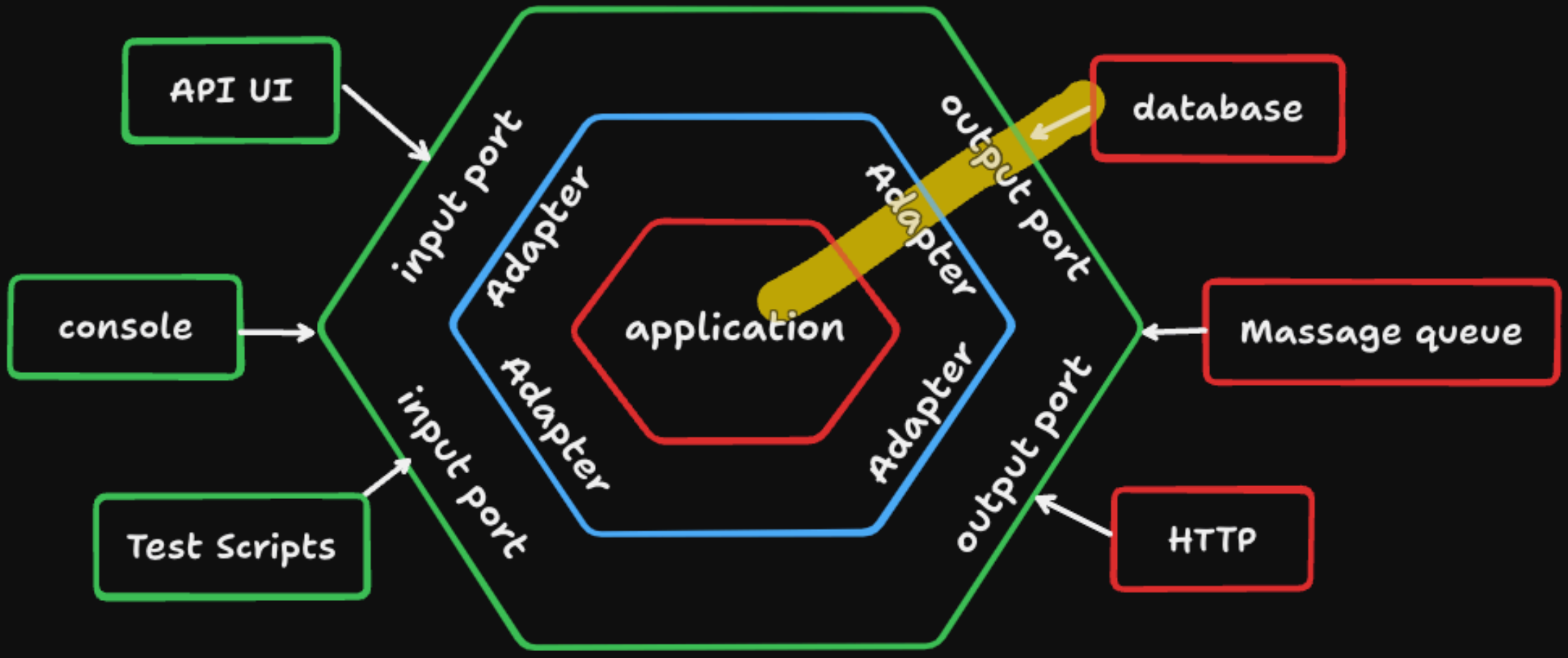
```ts
const userRepository = new UserRepositoryMemoryAdapter()
const userService = new UserService(userRepository)

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

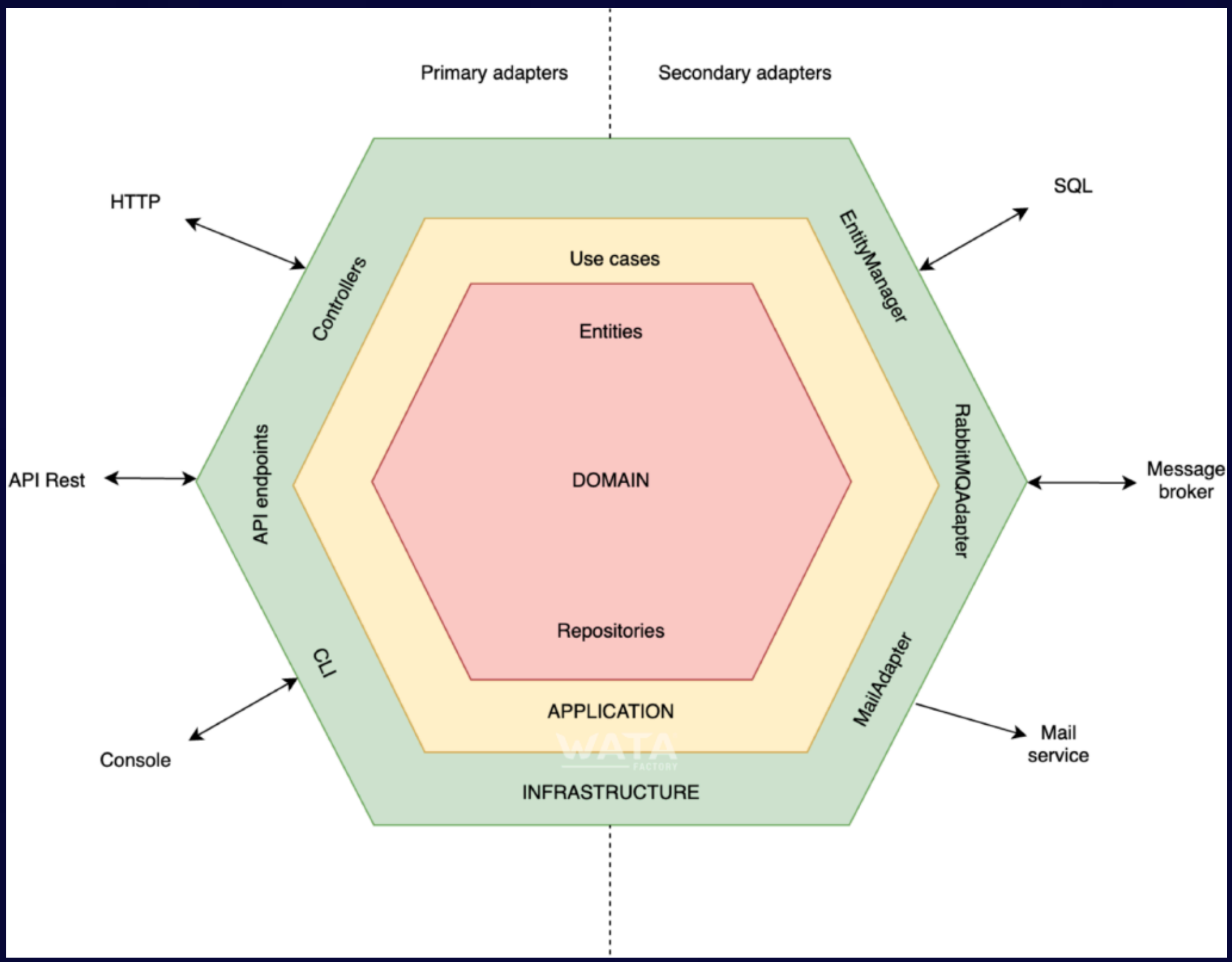
TS

- *agora as peças fazem mais sentido*



Isaac Gomes

- ***estrutura de pastas comum***

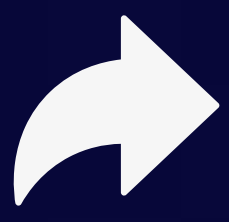


os nomes mais comuns são os acima, porem, vale lembrar que não tem uma nomenclatura obrigatória

Isaac Gomes

Gostou?

Curta



Compartilhe

Salve

Isaac Gomes