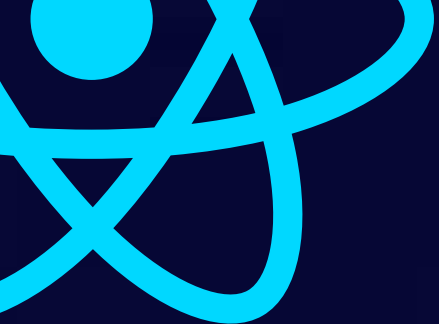


COMO CONSTRUIR PAGINAS FÁCEIS DE TESTAR REACT.JS

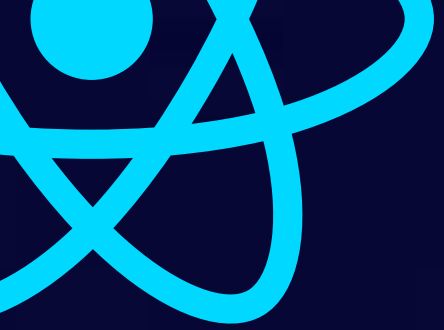


```
export const ListTaskPage = () => {  
  const http = new HttpClient()  
  const service: TaskServiceRegistry = {  
    createTask: new CreateTaskService(http),  
    deleteTask: new DeleteTaskService(http),  
    listTasks: new ListTaskService(http),  
  }  
  const methods = useListTaskModel(service)  
  return <ListTaskView {...methods} />  
}
```



Em React.js, é comum que a camada de apresentação acumule funções como serviços, schemas, requisições HTTP e controle de estados.





Esse tipo de arquitetura acaba gerando mocks excessivos, que muitas vezes mais parecem um verdadeiro assassinato do código



```
jest.mock('axios');
```

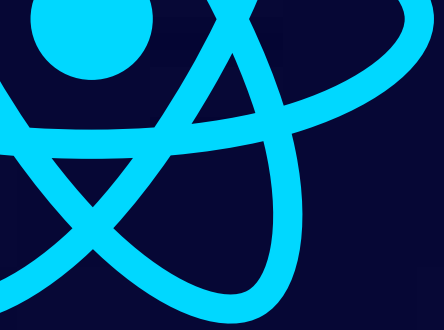
```
const mockData = { id: 1, name: 'Test' };  
(axios.get as jest.Mock)  
  .mockResolvedValue({ data: mockData });
```

```
const data = await fetchData('/api/test');
```

```
expect(data).toEqual(mockData);
```

```
expect(axios.get)
```

```
  .toHaveBeenCalledWith('/api/test');
```



COMO PODEMOS SOLUCIONAR ESSE TIPO DE PROBLEMA?

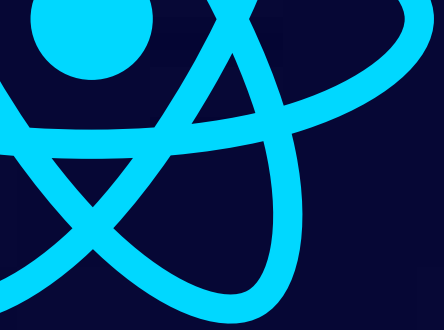


```
jest.mock('axios');

const mockData = { id: 1, name: 'Test' };
(axios.get as jest.Mock)
  .mockResolvedValue({ data: mockData });

const data = await fetchData('/api/test');

expect(data).toEqual(mockData);
expect(axios.get)
  .toHaveBeenCalledWith('/api/test');
```



PODEMOS DESENVOLVER ARQUITETURAS BASEADAS NA INVERSÃO DE DEPENDÊNCIAS.

Substituímos a dependência direta
do recurso por uma dependência
de uma interface.



```
export type TaskServiceRegistry = {  
  createTask: CreateTaskServiceContract  
  deleteTask: DeleteTaskServiceContract  
  listTasks: ListTaskServiceContract  
}
```

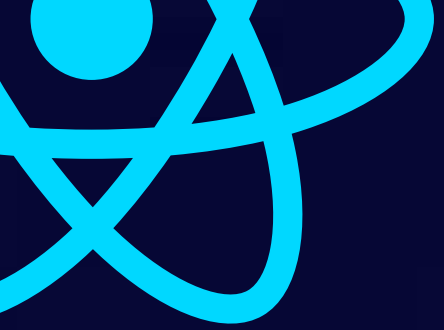


AGORA, OS SERVIÇOS ESTÃO SEPARADOS DAS PÁGINAS.

Em seguida, podemos isolar nossa camada de HTTP Criando Camadas na nossa arquitetura



```
export class CreateTaskService {  
  constructor(private http: HttpClient) {}  
  async exec(body: CreateTaskBody): Promise<Task> {  
    return await this.http.sendRequest({  
      endpoint: '/task/',  
      method: HttpMethod.POST,  
      body,  
    })  
  }  
}
```

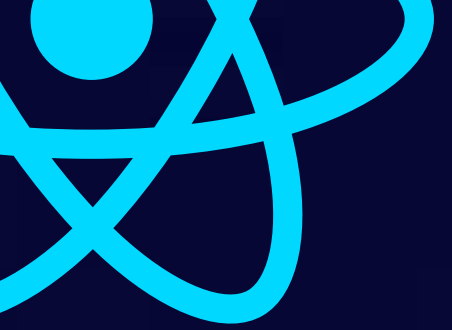



AGORA, CRIAMOS UMA INTERFACE PARA INTEGRAR NOSSA CAMADA DE HTTP.

Agora, nossos serviços
utilizam uma abstração.



```
export interface IHttpClient {  
  sendRequest: <TResponse, TBody = unknown>(  
    request: HttpRequest<TBody>,  
  ) => Promise<TResponse>  
}
```

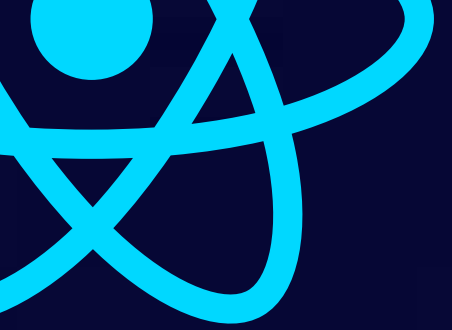


AGORA, PODEMOS CONSIDERAR UMA DIVISÃO

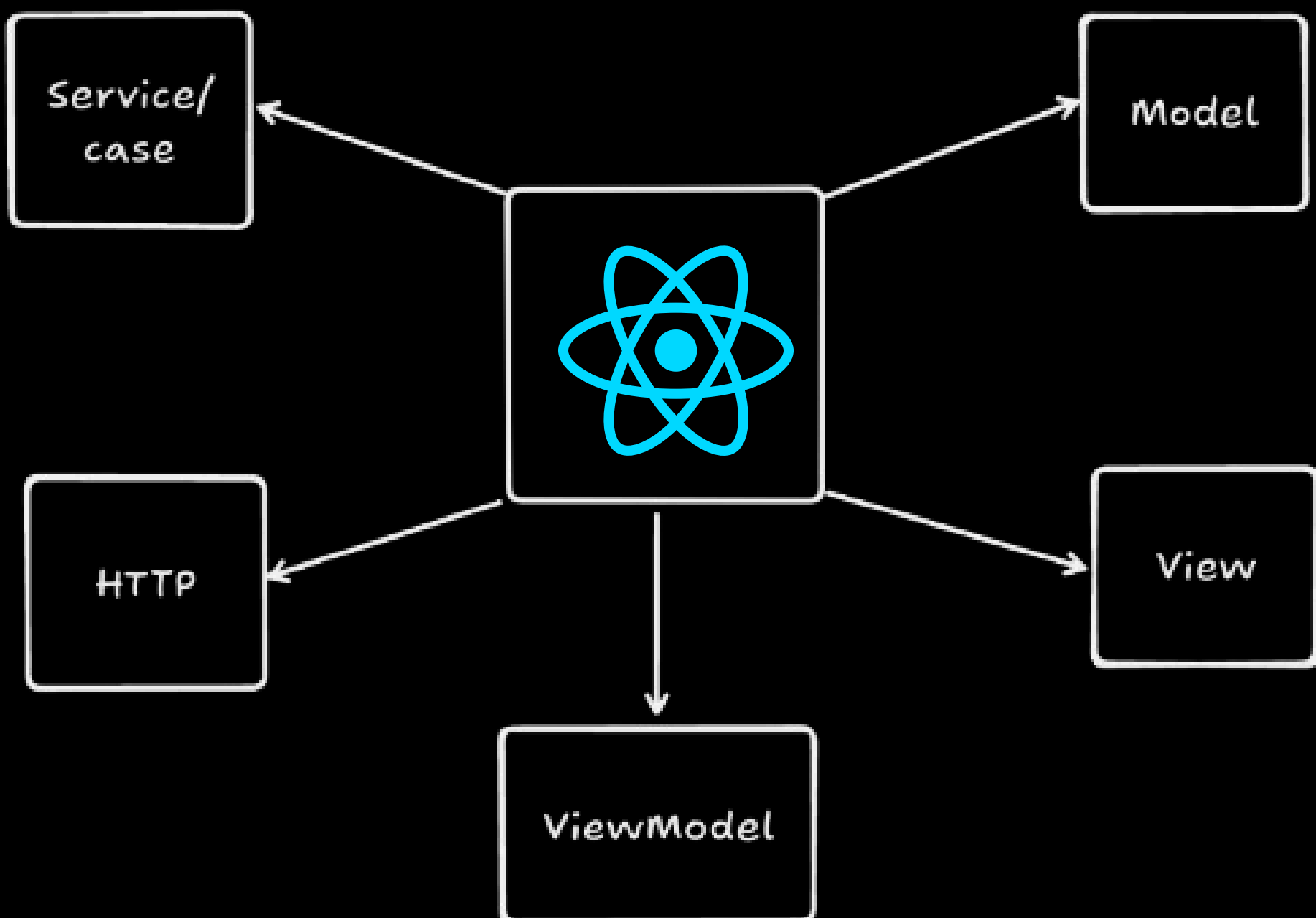
- Model (lógica),
- View (visual)
- Service (requisição)
- viewModel (camada de controle)

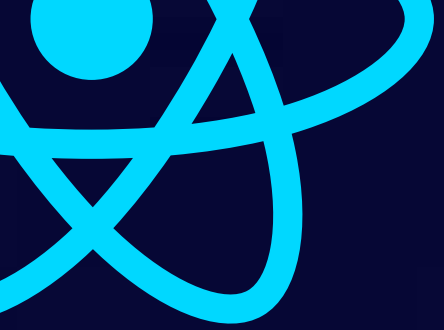


```
export const ListTaskPage = () => {  
  const http = new HttpClient()  
  const service: TaskServiceRegistry = {  
    createTask: new CreateTaskService(http),  
    deleteTask: new DeleteTaskService(http),  
    listTasks: new ListTaskService(http),  
  }  
  const methods = useListTaskModel(service)  
  return <ListTaskView {...methods} />  
}
```

**AGORA, TEMOS CAMADAS
MUITO MAIS CLARAS PARA A
REALIZAÇÃO DE TESTES.**

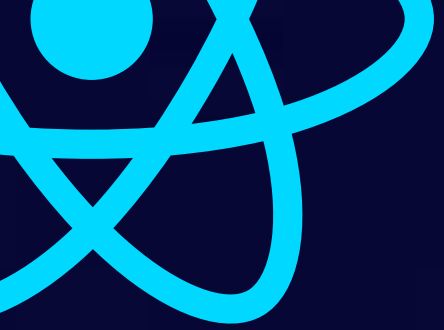




**AGORA, PODEMOS
SUBSTITUIR NOSSA CAMADA
DE SERVIÇO POR UM SERVIÇO
EM MEMÓRIA, PERMITINDO
TESTES SEM DEPENDÊNCIAS.**



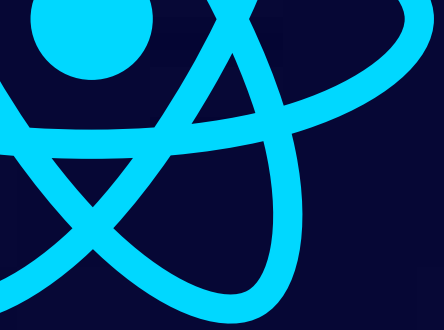
```
export const MakeSut = () => {  
  const methods = useListTaskModel({  
    createTask: new InMemoryCreateTaskService(),  
    deleteTask: new InMemoryDeleteTaskService(),  
    listTasks: new InMemoryListTaskService(),  
  })  
  return <ListTaskView {...methods} />  
}
```



**AGORA, PODEMOS
SUBSTITUIR NOSSA CAMADA
DE SERVIÇO POR UM SERVIÇO
EM MEMÓRIA, PERMITINDO
TESTES SEM DEPENDÊNCIAS.**



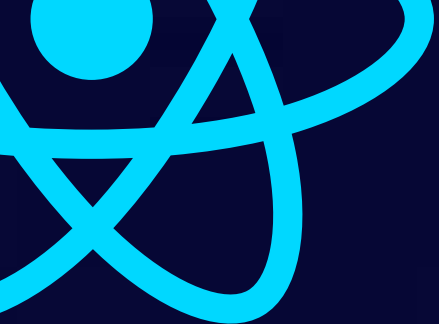
```
export const MakeSut = () => {  
  const methods = useListTaskModel({  
    createTask: new InMemoryCreateTaskService(),  
    deleteTask: new InMemoryDeleteTaskService(),  
    listTasks: new InMemoryListTaskService(),  
  })  
  return <ListTaskView {...methods} />  
}
```



PRONTO! CONSEGUIMOS TESTAR SEM PRECISAR FAZER MALABARISMOS.



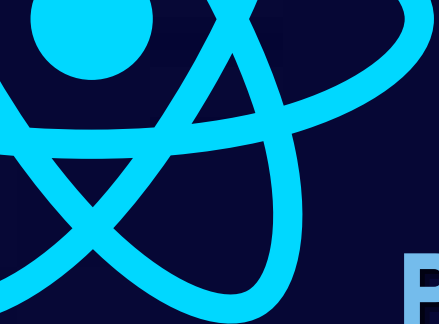
```
it('should add a task when the form is submitted', async () => {  
  const screen = renderView(<MakeSut />)  
  
  const input = screen.getByPlaceholderText('Digite uma tarefa')  
  fireEvent.change(input, { target: { value: 'Nova Tarefa' } })  
  
  const button = screen.getByText('Adicionar')  
  fireEvent.click(button)  
  
  await waitFor(() => {  
    expect(screen.getByText('Nova Tarefa')).toBeInTheDocument()  
  })  
})
```



ASSIM, FICA MUITO MAIS
FÁCIL ALCANÇAR UMA
ALTA COBERTURA.



File	% Stmts	% Branch	% Funcs
All files	100	100	100
data/task-service/create-task-service	100	100	100
create-task-service.spec.ts	100	100	100
create-task-service.ts	100	100	100
data/task-service/delete-task-service	100	100	100
delete-task-service.spec.ts	100	100	100
delete-task-service.ts	100	100	100
data/task-service/list-tasks-services	100	100	100
list-tasks-services.spec.ts	100	100	100
list-tasks-services.ts	100	100	100
infrastructure/contracts	100	100	100
http-contracts.ts	100	100	100
infrastructure/http	100	100	100
http-client.spec.ts	100	100	100
http-client.ts	100	100	100
presentation/Provider	100	100	100
ProviderSystem.tsx	100	100	100
presentation/pages/list-task	100	100	100
list-task-model.ts	100	100	100
list-task-view.tsx	100	100	100
list-task.spec.tsx	100	100	100

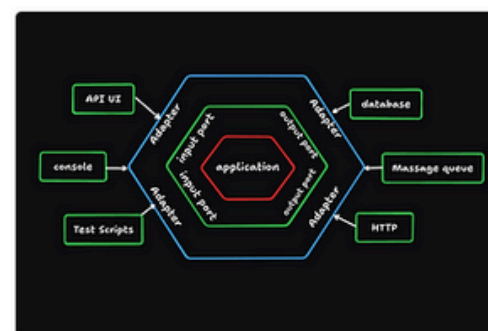



PARA MAIS DETALHES SOBRE O CÓDIGO, CONSULTE OS ÚLTIMOS ARTIGOS

Arquitetura Front-end: Abordagens Hexagonal, Adapter, Inversão de Dependências e Testes Unitários

Uma das situações mais comuns no desenvolvimento de software é o uso excessivo de siglas, como DDD, TDD e SOLID....

Oct 11 🖱️ 23




 Isaac Gomes

Arquitetura Front-end: Camadas Eficientes em React.js

Camadas

Oct 8 🖱️ 3

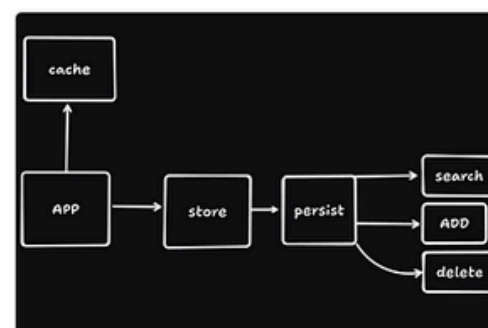


 Isaac Gomes

Arquitetura Front-end: Planejando o Projeto React.js

um dos problemas clássicos em arquitetura Frontend é ausência do planejamento da criação dos app.

Sep 29 🖱️ 18 💬 1



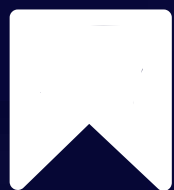
Gostou?



Curta



Compartilhe



Salve