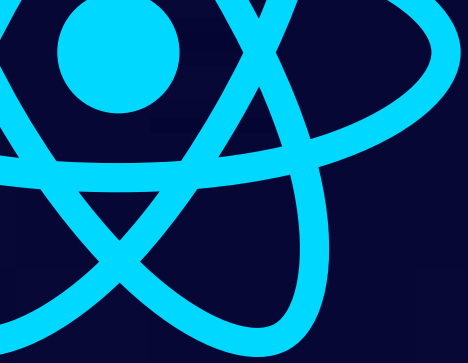


# COMO FAZER INJEÇÃO DE DEPENDÊNCIAS NO REACT.JS



```
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} />  
}
```

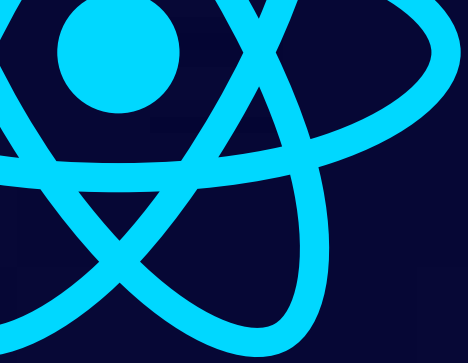


# NO ENTANTO, OBSERVE QUE ESSA ABORDAGEM DE INJEÇÃO ACABA NÃO SENDO VIÁVEL

Estamos injetando manualmente as  
requisições, o que torna a gestão  
difícil em páginas complexas.



```
const http = new HttpClient()  
const service: TaskServiceRegistry = {  
  createTask: new CreateTaskService(http),  
  deleteTask: new DeleteTaskService(http),  
  listTasks: new ListTaskService(http),  
}
```

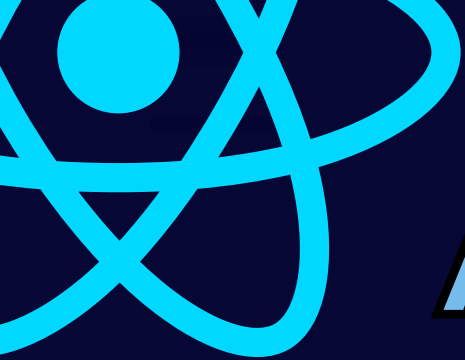


**ISSO CRIA A NECESSIDADE DE GERENCIAR ESSA DEPENDÊNCIA DE FORMA ACESSÍVEL EM DIFERENTES PARTES DA PÁGINA.**

Para isso, criaremos um gerenciador de dependências usando o context

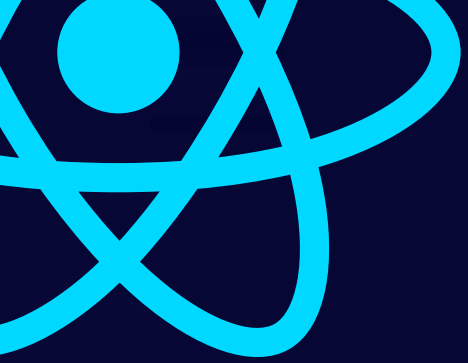


```
const TodoListProvider = (props) => {  
  const { children, services } = props  
  return (  
    <TodoListContext.Provider value={{...services}}>  
      {children}  
    </TodoListContext.Provider>  
  )  
}
```



# AGORA NOSSA VIEWMODEL INJETA AS DEPENDÊNCIAS AUTOMATICAMENTE VIA CONTEXT.

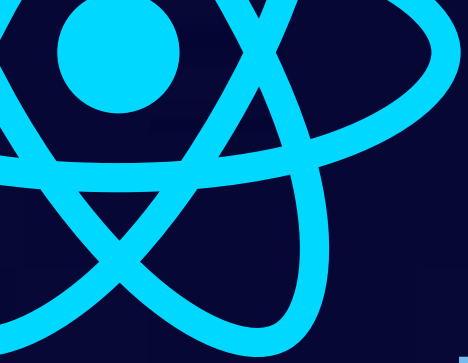
```
const ListTaskViewModel = () => {  
  const http = new HttpClient()  
  const created = new CreateTaskService(http)  
  const deleted = new DeleteTaskService(http)  
  const list = new ListTaskService(http)  
  
  return (  
    <TodoListProvider  
      services={{ created,deleted,list }}  
    >  
      <ListTaskView />  
    </TodoListProvider>  
  )  
}
```



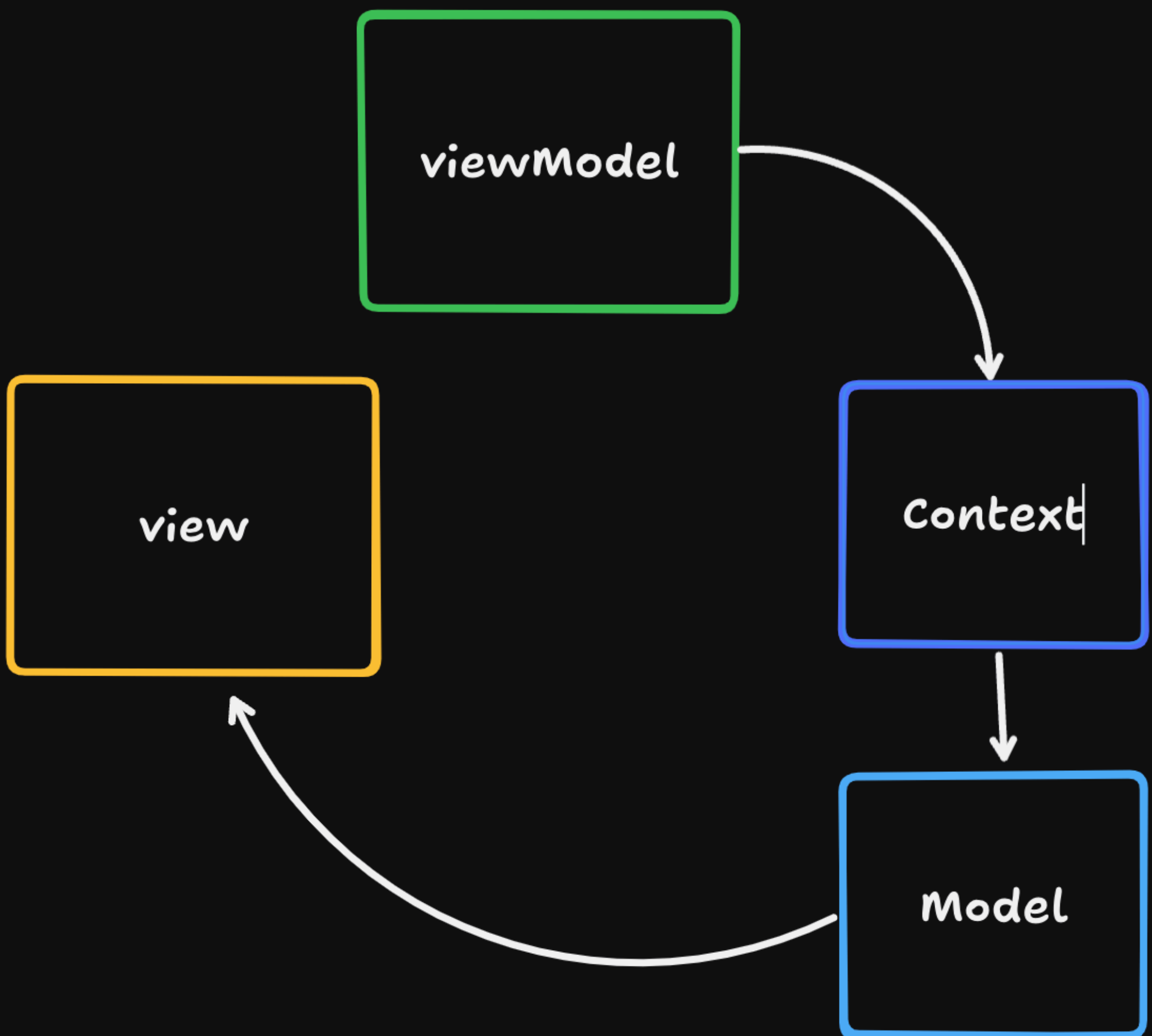
# AGORA PODEMOS ACESSAR NOSSAS DEPENDÊNCIAS DIRETAMENTE A PARTIR DO CONTEXT.

Preservamos a flexibilidade de  
nossos serviços por meio da  
inversão das dependências já  
estabelecidas.

```
const useListTaskModel = () => {  
  const { created, deleted, list } =  
    useTodoListContainer()  
  
  return {}  
}
```



Nossa ViewModel injeta dependências via Context, facilitando o acesso direto no Model, que é chamado na View.





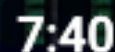


12:21

•

•

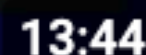
•



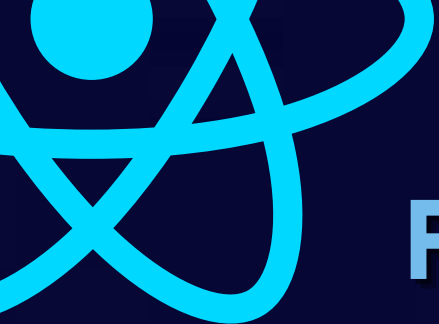
● ● ●



•  
•  
•



•

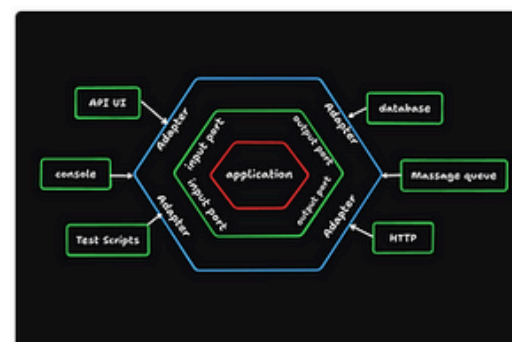



# PARA MAIS DETALHES SOBRE O CÓDIGO, CONSULTE OS ÚLTIMOS ARTIGOS

## 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

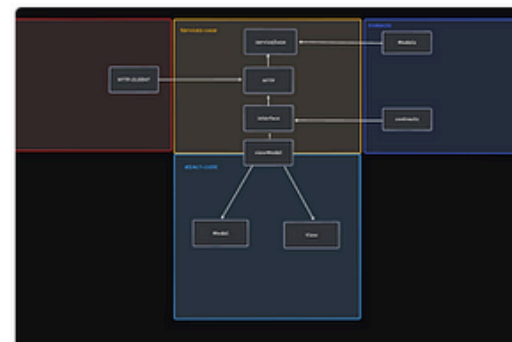



 Isaac Gomes

## Arquitetura Front-end: Camadas Eficientes em React.js

Camadas

Oct 8 🖱️ 3

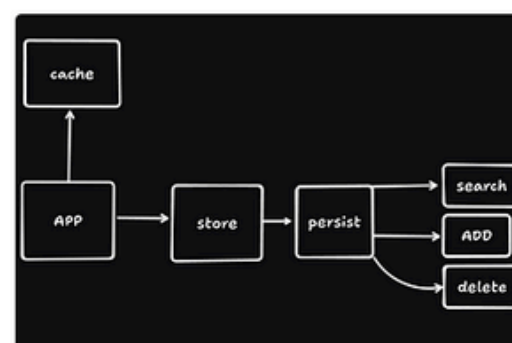


 Isaac Gomes

## 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





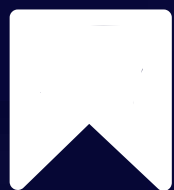
# Gostou?



**Curta**



**Compartilhe**



**Salve**