Crie Funções melhores em Typescript

```
ParseFormatUser(
        user.firstName,
        user.lastName,
        user.streetAddress,
        user.postalCode
)
```

```
parseFormatUser({
        firstName: "React",
        lastName: "typescript",
        streetAddress: "200",
        postalCode: "300",
})
```

Isaac Gomes

TS

IntelliSense

Quando criamos funções com muitos parâmetros acabamos com alguns problemas

```
const parseFormatUser = (
    firstName: string,
    lastName: string,
    streetAddress: string,
    postalCode: string
) => {}
```

Isaac Gomes

TS

IntelliSense

Como: ordem, dificuldade para ler e a perca do IntelliSense

```
ParseFormatUser(
    user.firstName,
    user.lastName,
    user.streetAddress,
    user.postalCode
)
```

Isaac Gomes

TS

IntelliSense

para a ordem parar de importar
podemos usar um Objeto
tornando mais legível

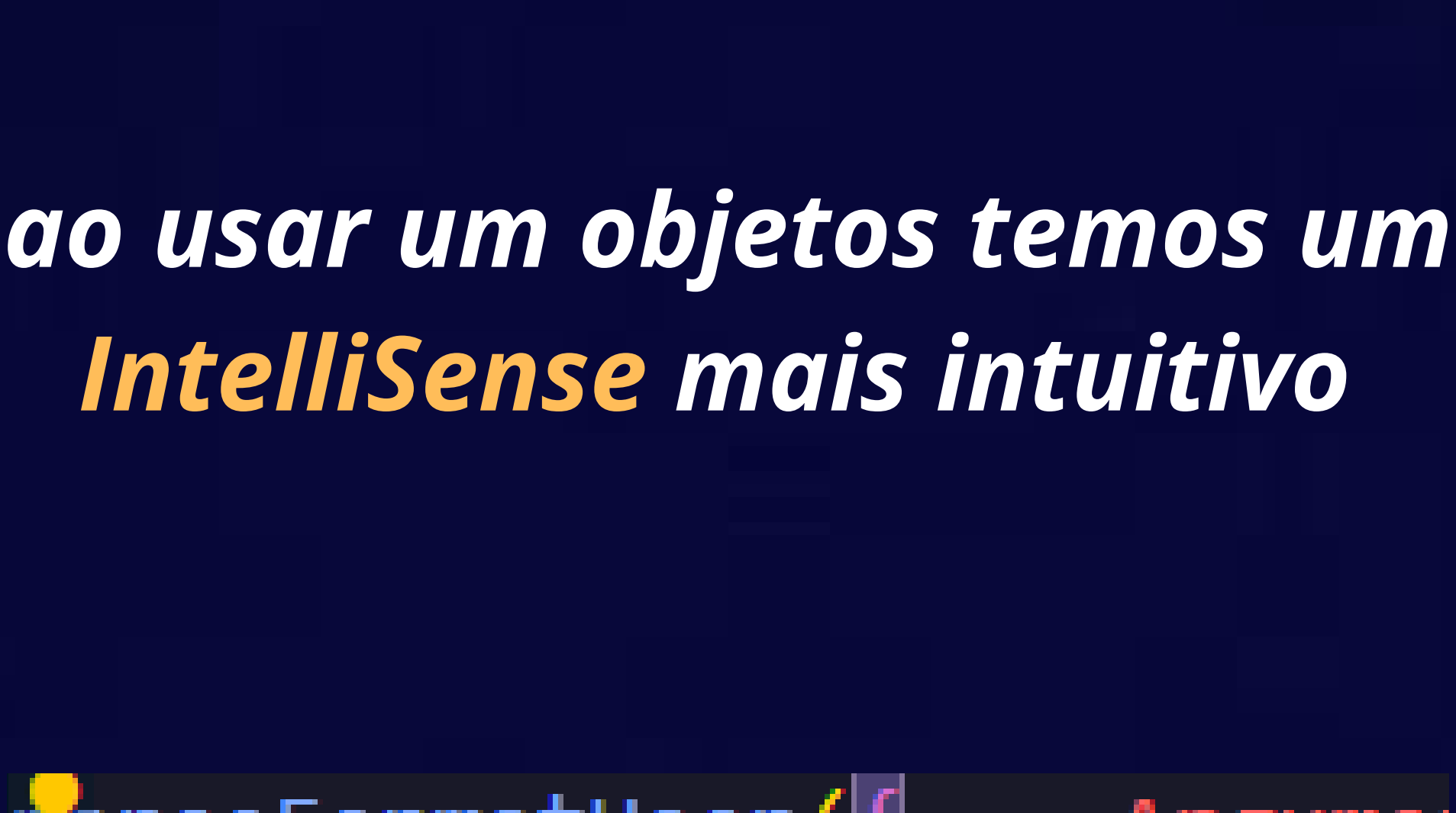
```
const parseFormatUser = ({
      firstName,
      lastName,
      postalCode,
      streetAddress
}: User) => {}
```

Isaac Gomes

TS

IntelliSense

ao usar um objetos temos um IntelliSense mais intuitivo



Isaac Gomes

TS

IntelliSense

retorna Array pode ser uma péssima ideia pois você cria vários problemas

```
const useCountdown = (date: string) => {
    return [
        day,
        hour,
        minute,
        second
    ]
}
```

Isaac Gomes

TS

IntelliSense

podemos desestruturar com nomes diferentes

```
const [ a, b, c, d ] = useCountdown(
'Jan 1, 2025, 00:00:00'
)
```

Isaac Gomes

TS

🧠 IntelliSense

quando usamos um array perdemos o IntelliSense e podemos retirar até mais itens do que é retornados

```
const [ a, b, c, d,e ] = useCountdown(
'Jan 1, 2025, 00:00:00'
)
```

Isaac Gomes

TS

IntelliSense

Ja quando usamos um objetos temos o IntelliSense e usamos o mesmo nomes

```ts
const useCountdown = (date: string) => {
    return {day, hour, minute, second}
}

const {
  day,
  hour,
  minute,
  second
} = useCountdown('Jan 1, 2025, 00:00:00')
```

Isaac Gomes

🧠 IntelliSense

quando criamos uma função que recebe uma string pode ser qualquer string no entanto em algumas funções não é o comportamento que esperamos

```ts
const handleStatus = (type: string): string => {
  if (type === 'APPROVED') return 'aprovado'
  if (type === 'REJECTED') return 'reprovado'
  return 'ajuste'
}

handleStatus("")
handleStatus("a")
handleStatus("APPROVED")
handleStatus('REJECTED')
handleStatus('ADJUSTED')
```

Isaac Gomes

TS

🧠 IntelliSense

para isso podemos mapear nossos casos em objetos deixando nossas funções mais simples

```ts
const status = {
  APPROVED: 'aprovado',
  REJECTED: 'reprovado',
  ADJUSTED: 'ajuste',
} as const

const handleStatus = (type: StatusActions)=>{
  return status[type]
}

handleStatus("")
handleStatus("a")
//Argument of type '""'
//is not assignable to parameter of type
//'"APPROVED" | "REJECTED" | "ADJUSTED"'.
handleStatus("ADJUSTED")
handleStatus("APPROVED")
handleStatus("REJECTED")
```

Isaac Gomes

TS

🧠 IntelliSense

saiba o momento de quebrar uma função para separar as responsabilidades e deixar fácil de ler

```ts
const handleCreateStudent = async (student: CreateStudent) => {
  if (
    student.studyMaterial.length > 0 &&
    student.age > 18
  ) return

  const format = {
    student: {
      name: student.name,
      age: student.age,
      document: student.document,
    },
    faculty: {
      collegeCampus: student.collegeCampus,
      studyMaterial: student.studyMaterial,
    },
  }

  try { const { data } = await axios.post('/', format) }
  catch (err) {console.log(err)}
}
```

Isaac Gomes

TS

🧠 IntelliSense

quando quebramos nossas responsabilidades tornamos nossos trechos de funções nomeados... logo fica simples de saber oq cada parte faz

```ts
const handleCreateStudent = async (student: CreateStudent) => {
  if (isEligibleForRegistration(student)) return
  const formattedStudentData = formatStudentData(student)
  await CreateStudentService(formattedStudentData)
}
```

Isaac Gomes

TS

IntelliSense

evite deixar strings soltas no código elas geralmente só fazem sentido no momento da construção depois sem o contexto leva muito tempo para descobrir o que elas significam

```ts
export const tasksReducer = (
  state: Task[],
  action: Action)
  : Task[] =>
{
  switch (action.type) {
    case 'ADD_TASK':
      return [...state, action.task]
    case 'DELETE_TASK':
      return state.filter(task => task.id !== action.taskId)
    default:
      return state
  }
}
```

Isaac Gomes

TS

🧠 IntelliSense

*para melhorar o entendimento podemos usar **enum** tornando nossas **strings** algo nomeados em um **contexto***

```ts
export const tasksReducer = (
  state: Task[],
  action: Action
): Task[] => {
  switch (action.type) {
    case ActionTypes.add:
      return [...state, action.task]
    case ActionTypes.delete:
      return state.filter(task => task.id !== action.taskId)
    default:
      return state
  }
}
```

Isaac Gomes

TS

Gostou?

Curta

Compartilhe

Salve

Isaac Gomes