

```

type CustomValidator<T> = (value: T[keyof T]) => void | NodeOrString;
export enum DefaultValidators {
  Email = 'email',
  Required = 'required',
}
export type ValidationMap<T> = {
  [K in keyof T]?:
    | CustomValidator<T>
    | CustomValidator<T>[]
    | DefaultValidators
    | DefaultValidators[]
};
export type ValidatorErrors<T> = { [K in keyof T]?: NodeOrString };
interface ReduceValue<T> {
  errors: ValidatorErrors<T>;
  hasErrors: boolean;
}

const setError = <T>(<
  previousErrors: ReduceValue<T>,
  key: keyof T,
  error: void | NodeOrString,
): ReduceValue<T> =>
  error
  ? {
    errors: Object.assign({}, previousErrors.errors, { [key]: error },
    hasErrors: true,
  }
  : previousErrors;

const isRequired = <V>(value: V) => !value && Translate.translate('isRequired');

const validateEmail = <V>(value: V) =>
  typeof value === 'string' &&
  !isEmail(value) &&
  Translate.translate('isEmail');

const validateSingleKey = <T extends object, K extends keyof T>(<
  value: T[K]

```

# TYPES

# KEEPING YOUR CODE SAFE AND WARM

**TYPES ARE A LOT OF WORK... I DON'T  
WANT TO ADD MORE.**

**A lot of people**