



LA RESERVA - CONSTANTES Y CONFIGURACIÓN

PARTE 5: Validadores y Formateadores

Versión: 1.0

Fecha: Octubre 2025



ARCHIVO: validators.ts

de: `src/utils/validators.ts`

```
// src/utils/validators.ts

/**
 * LA RESERVA - VALIDADORES CON ZOD
 *
 * Schemas de validación para formularios usando Zod.
 * Proporciona validación type-safe en cliente y servidor.
 */

import { z } from 'zod';
import { VALIDATION, ERROR_MESSAGES } from '../constants';

// =====
// 1. SCHEMA: COTIZACIÓN
// =====

export const quoteSchema = z.object({
  name: z
    .string()
    .min(VALIDATION.name.min, ERROR_MESSAGES.minLength(VALIDATION.name.min))
    .max(VALIDATION.name.max, ERROR_MESSAGES.maxLength(VALIDATION.name.max))

  email: z
```

```

        .string()
        .email(ERROR_MESSAGES.invalidEmail)
        .max(VALIDATION.email.max),

phone: z
    .string()
    .min(VALIDATION.phone.min, ERROR_MESSAGES.minLength(VALIDATION.phone.
    .max(VALIDATION.phone.max, ERROR_MESSAGES.maxLength(VALIDATION.phone.
    .regex(/^(\\+?51)?9\\d{8}$/, ERROR_MESSAGES.invalidPhone),

eventType: z
    .string()
    .min(1, ERROR_MESSAGES.required),

eventDate: z
    .string()
    .min(1, ERROR_MESSAGES.required)
    .refine((date) => {
        const eventDate = new Date(date);
        const today = new Date();
        today.setHours(0, 0, 0, 0);
        return eventDate >= today;
    }, ERROR_MESSAGES.pastDate),

guestCount: z
    .number()
    .min(VALIDATION.guests.min, ERROR_MESSAGES.minValue(VALIDATION.guests
    .max(VALIDATION.guests.max, ERROR_MESSAGES.maxValue(VALIDATION.guests

message: z
    .string()
    .min(VALIDATION.message.min, ERROR_MESSAGES.minLength(VALIDATION.mess
    .max(VALIDATION.message.max, ERROR_MESSAGES.maxLength(VALIDATION.mess
    .optional(),
));

export type QuoteFormData = z.infer<typeof quoteSchema>;

// =====
// 2. SCHEMA: CONTACTO
// =====

export const contactSchema = z.object({
    name: z

```

```

        .string()
        .min(VALIDATION.name.min, ERROR_MESSAGES.minLength(VALIDATION.name.mi
        .max(VALIDATION.name.max, ERROR_MESSAGES.maxLength(VALIDATION.name.ma

email: z
    .string()
    .email(ERROR_MESSAGES.invalidEmail),

phone: z
    .string()
    .optional(),

subject: z
    .string()
    .min(3, ERROR_MESSAGES.minLength(3))
    .max(200, ERROR_MESSAGES.maxLength(200)),

message: z
    .string()
    .min(VALIDATION.message.min, ERROR_MESSAGES.minLength(VALIDATION.mess
    .max(VALIDATION.message.max, ERROR_MESSAGES.maxLength(VALIDATION.mess
});

export type ContactFormData = z.infer<typeof contactSchema>;

// =====
// 3. SCHEMA: NEWSLETTER
// =====

export const newsletterSchema = z.object({
    email: z
        .string()
        .email(ERROR_MESSAGES.invalidEmail),
});

export type NewsletterFormData = z.infer<typeof newsletterSchema>;

// =====
// 4. SCHEMA: LOGIN ADMIN
// =====

export const loginSchema = z.object({
    email: z
        .string()

```

```

        .email(ERROR_MESSAGES.invalidEmail),

password: z
    .string()
    .min(6, ERROR_MESSAGES.minLength(6)),
});

export type LoginFormData = z.infer<typeof loginSchema>;

// =====
// 5. SCHEMA: EVENTO (Admin)
// =====

export const eventSchema = z.object({
    clientName: z
        .string()
        .min(VALIDATION.name.min),

    clientEmail: z
        .string()
        .email(),

    clientPhone: z
        .string()
        .min(VALIDATION.phone.min),

    eventType: z
        .string()
        .min(1),

    eventDate: z
        .string()
        .min(1),

    eventTime: z
        .string()
        .optional(),

    guestCount: z
        .number()
        .min(VALIDATION.guests.min)
        .max(VALIDATION.guests.max),

    venue: z

```

```

        .string()
        .optional(),

    packageId: z
        .string()
        .optional(),

    totalPrice: z
        .number()
        .min(0)
        .optional(),

    deposit: z
        .number()
        .min(0)
        .optional(),

    status: z
        .enum(['pending', 'confirmed', 'completed', 'cancelled']),

    notes: z
        .string()
        .optional(),
  });

export type EventFormData = z.infer<typeof eventSchema>;

```



ARCHIVO: formatters.ts

de: src/utils/formatters.ts

```

// src/utils/formatters.ts

/**
 * LA RESERVA - FUNCIONES DE FORMATEO
 *
 * Funciones para formatear datos (fechas, moneda, números, etc.)
 */

import { format, formatDistanceToNow, isToday, isYesterday, isTomorrow }
import { es } from 'date-fns/locale';

```

```
// =====
// 1. FORMATEO DE FECHAS
// =====

/**
 * Formatea una fecha a formato legible en español
 *
 * @param date - Fecha a formatear
 * @param formatStr - Formato deseado (default: 'PPP')
 * @returns Fecha formateada
 *
 * @example
 * formatDate(new Date()) → '22 de octubre de 2025'
 */
export function formatDate(date: Date | string, formatStr: string = 'PPP') {
  const dateObj = typeof date === 'string' ? new Date(date) : date;
  return format(dateObj, formatStr, { locale: es });
}

/**
 * Formatea una fecha de manera relativa
 *
 * @param date - Fecha a formatear
 * @returns Fecha relativa
 *
 * @example
 * formatRelativeDate(yesterday) → 'hace 1 día'
 */
export function formatRelativeDate(date: Date | string): string {
  const dateObj = typeof date === 'string' ? new Date(date) : date;

  if (isToday(dateObj)) return 'Hoy';
  if (isYesterday(dateObj)) return 'Ayer';
  if (isTomorrow(dateObj)) return 'Mañana';

  return formatDistanceToNow(dateObj, { addSuffix: true, locale: es });
}

/**
 * Formatea hora a formato 12h
 *
 * @param time - Hora en formato 24h ('14:30')
 * @returns Hora en formato 12h
 */

```

```

*
* @example
* formatTime('14:30') → '2:30 PM'
*/
export function formatTime(time: string): string {
  const [hours, minutes] = time.split(':').map(Number);
  const period = hours >= 12 ? 'PM' : 'AM';
  const displayHours = hours % 12 || 12;
  const displayMinutes = minutes.toString().padStart(2, '0');

  return `${displayHours}:${displayMinutes} ${period}`;
}

// =====
// 2. FORMATEO DE MONEDA
// =====

/**
 * Formatea moneda en soles peruanos
 *
 * @param amount - Cantidad a formatear
 * @returns Moneda formateada
 *
 * @example
 * formatCurrency(1500) → 'S/ 1,500'
 */
export function formatCurrency(amount: number): string {
  return new Intl.NumberFormat('es-PE', {
    style: 'currency',
    currency: 'PEN',
    minimumFractionDigits: 0,
    maximumFractionDigits: 0,
  }).format(amount);
}

/**
 * Formatea moneda con decimales
 *
 * @param amount - Cantidad a formatear
 * @returns Moneda formateada con decimales
 *
 * @example
 * formatCurrencyWithDecimals(1500.50) → 'S/ 1,500.50'
 */

```

```

export function formatCurrencyWithDecimals(amount: number): string {
  return new Intl.NumberFormat('es-PE', {
    style: 'currency',
    currency: 'PEN',
    minimumFractionDigits: 2,
    maximumFractionDigits: 2,
  }).format(amount);
}

```

```

// =====
// 3. FORMATEO DE NÚMEROS
// =====

```

```

/**
 * Formatea número con separadores de miles
 *
 * @param num - Número a formatear
 * @returns Número formateado
 *
 * @example
 * formatNumber(1500) → '1,500'
 */
export function formatNumber(num: number): string {
  return new Intl.NumberFormat('es-PE').format(num);
}

```

```

/**
 * Formatea porcentaje
 *
 * @param value - Valor decimal (0.15 = 15%)
 * @returns Porcentaje formateado
 *
 * @example
 * formatPercentage(0.15) → '15%'
 */
export function formatPercentage(value: number): string {
  return `${Math.round(value * 100)}%`;
}

```

```

// =====
// 4. FORMATEO DE TEXTO
// =====

```

```

/**

```

```

* Formatea un nombre completo
*
* @param name - Nombre a formatear
* @returns Nombre capitalizado
*
* @example
* formatName('juan p  rez') → 'Juan P  rez'
*/
export function formatName(name: string): string {
  return name
    .toLowerCase()
    .split(' ')
    .map(word => word.charAt(0).toUpperCase() + word.slice(1))
    .join(' ');
}

/**
* Formatea duraci  n en horas
*
* @param hours - N  mero de horas
* @returns Duraci  n formateada
*
* @example
* formatDuration(5) → '5 horas'
*/
export function formatDuration(hours: number): string {
  if (hours === 1) return '1 hora';
  return `${hours} horas`;
}

/**
* Formatea rango de invitados
*
* @param range - Rango en formato '100-200'
* @returns Rango formateado
*
* @example
* formatGuestRange('100-200') → '100 - 200 invitados'
*/
export function formatGuestRange(range: string): string {
  if (range.includes('+')) {
    return `M  s de ${range.replace('+', ' ')} invitados`;
  }
  return `${range.replace('-', ' - ')} invitados`;
}

```

```

}

// =====
// 5. FORMATEO DE TELÉFONO
// =====

/**
 * Formatea número de teléfono peruano
 *
 * @param phone - Teléfono a formatear
 * @returns Teléfono formateado
 *
 * @example
 * formatPhoneNumber('999888777') → '+51 999 888 777'
 */
export function formatPhoneNumber(phone: string): string {
  const cleaned = phone.replace(/\D/g, '');
  const withoutCountryCode = cleaned.startsWith('51')
    ? cleaned.slice(2)
    : cleaned;

  if (withoutCountryCode.length === 9) {
    return `+51 ${withoutCountryCode.slice(0, 3)} ${withoutCountryCode.sl
  }

  return phone;
}

// =====
// 6. FORMATEO DE ARCHIVOS
// =====

/**
 * Formatea tamaño de archivo
 *
 * @param bytes - Tamaño en bytes
 * @returns Tamaño formateado
 *
 * @example
 * formatFileSize(1024) → '1 KB'
 */
export function formatFileSize(bytes: number): string {
  if (bytes === 0) return '0 Bytes';

```

```
const k = 1024;
const sizes = ['Bytes', 'KB', 'MB', 'GB'];
const i = Math.floor(Math.log(bytes) / Math.log(k));

return `${Math.round(bytes / Math.pow(k, i))} ${sizes[i]}`;
}

/**
 * FIN DE LA PARTE 5
 * Continúa en: 06F-CONSTANTES-Y-CONFIG-Parte6.md
 */
```



VERIFICACIÓN - PARTE 5

- ☐ Schemas de Zod para todos los formularios
 - ☐ Validaciones correctas con mensajes de error
 - ☐ Formateo de fechas en español
 - ☐ Formato de moneda en soles
 - ☐ Formateo de números con separadores
 - ☐ Formateo de telefonos peruanos
-

Próximo archivo: Parte 6 - WhatsApp y Configuraciones Finales

© 2025 La Reserva