# 💻 LA RESERVA - CONSTANTES Y CONFIGURACIÓN

## PARTE 4: Funciones de Utilidad (utils.ts)

**Versión:** 1.0
**Fecha:** Octubre 2025

---

## 📄 ARCHIVO: utils.ts

### Ubicación: `src/utils/utils.ts`

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

/**
 * LA RESERVA - FUNCIONES DE UTILIDAD
 *
 * Funciones helper reutilizables en todo el proyecto.
 *
 * @version 1.0
 */

import { type ClassValue, clsx } from 'clsx';
import { twMerge } from 'tailwind-merge';

// ============================================
// 1. UTILIDADES DE ESTILO
// ============================================

/**
 * Combina clases de Tailwind sin conflictos
 *
 * @param inputs - Clases a combinar
 * @returns String con clases combinadas
 *
```

```
 * @example
 * cn('px-4', 'px-6') → 'px-6' (última tiene prioridad)
 * cn('btn', isActive && 'btn-active') → 'btn btn-active'
 */
export function cn(...inputs: ClassValue[]) {
  return twMerge(clsx(inputs));
}

// ============================================
// 2. UTILIDADES DE TEXTO
// ============================================

/**
 * Genera un slug URL-friendly desde un string
 *
 * @param text - Texto a convertir
 * @returns Slug en formato URL
 *
 * @example
 * slugify('Boda en Lima 2025') → 'boda-en-lima-2025'
 */
export function slugify(text: string): string {
  return text
    .toString()
    .toLowerCase()
    .normalize('NFD')
    .replace(/[\u0300-\u036f]/g, '')
    .replace(/[^a-z0-9\s-]/g, '')
    .trim()
    .replace(/\s+/g, '-')
    .replace(/-+/g, '-');
}

/**
 * Capitaliza la primera letra de cada palabra
 *
 * @param text - Texto a capitalizar
 * @returns Texto capitalizado
 *
 * @example
 * capitalize('juan pérez') → 'Juan Pérez'
 */
export function capitalize(text: string): string {
  return text
```

```
    .toLowerCase()
    .split(' ')
    .map(word => word.charAt(0).toUpperCase() + word.slice(1))
    .join(' ');
}

/**
 * Trunca un texto a cierta longitud agregando ellipsis
 *
 * @param text - Texto a truncar
 * @param length - Longitud máxima
 * @returns Texto truncado
 *
 * @example
 * truncate('Este es un texto muy largo', 10) → 'Este es un...'
 */
export function truncate(text: string, length: number): string {
  if (text.length <= length) return text;
  return text.slice(0, length).trim() + '...';
}

/**
 * Extrae iniciales de un nombre
 *
 * @param name - Nombre completo
 * @returns Iniciales
 *
 * @example
 * getInitials('Juan Pérez') → 'JP'
 */
export function getInitials(name: string): string {
  return name
    .split(' ')
    .map(word => word[0])
    .join('')
    .toUpperCase()
    .slice(0, 2);
}

// ============================================
// 3. VALIDACIONES
// ============================================

/**
```

```
 * Valida si un email es válido (básico)
 * Para validación robusta, usar Zod en validators.ts
 *
 * @param email - Email a validar
 * @returns true si es válido
 */
export function isValidEmail(email: string): boolean {
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return emailRegex.test(email);
}

/**
 * Valida si un teléfono peruano es válido
 * Formato: +51 seguido de 9 dígitos (9XXXXXXXX)
 *
 * @param phone - Teléfono a validar
 * @returns true si es válido
 */
export function isValidPeruvianPhone(phone: string): boolean {
  const phoneRegex = /^(\+?51)?9\d{8}$/;
  return phoneRegex.test(phone.replace(/\s/g, ''));
}

/**
 * Valida si una fecha es futura
 *
 * @param date - Fecha a validar
 * @returns true si es futura
 */
export function isFutureDate(date: Date | string): boolean {
  const dateObj = typeof date === 'string' ? new Date(date) : date;
  return dateObj > new Date();
}

// ============================================
// 4. FORMATEO
// ============================================

/**
 * Formatea un número de teléfono peruano
 *
 * @param phone - Teléfono a formatear
 * @returns Teléfono formateado
 *
```

```
 * @example
 * formatPhone('999888777') → '+51 999 888 777'
 */
export function formatPhone(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;
}

// ============================================
// 5. UTILIDADES DE FECHA
// ============================================

/**
 * Obtiene días entre dos fechas
 *
 * @param date1 - Primera fecha
 * @param date2 - Segunda fecha
 * @returns Número de días
 */
export function getDaysBetween(date1: Date, date2: Date): number {
  const oneDay = 24 * 60 * 60 * 1000;
  return Math.round(Math.abs((date1.getTime() - date2.getTime()) / oneDay
}

/**
 * Obtiene el saludo apropiado según la hora
 *
 * @returns Saludo del día
 */
export function getGreeting(): string {
  const hour = new Date().getHours();
  if (hour < 12) return 'Buenos días';
  if (hour < 19) return 'Buenas tardes';
  return 'Buenas noches';
}

// ============================================
```

```
// 6. UTILIDADES DEL DOM
// ============================================

/**
 * Scroll suave a un elemento
 *
 * @param elementId - ID del elemento
 * @param offset - Offset desde el top (default: 80px)
 */
export function scrollToElement(elementId: string, offset: number = 80):
  const element = document.getElementById(elementId);
  if (element) {
    const elementPosition = element.getBoundingClientRect().top;
    const offsetPosition = elementPosition + window.pageYOffset - offset;

    window.scrollTo({
      top: offsetPosition,
      behavior: 'smooth'
    });
  }
}

/**
 * Copia texto al clipboard
 *
 * @param text - Texto a copiar
 * @returns Promise con resultado
 */
export async function copyToClipboard(text: string): Promise<boolean> {
  try {
    await navigator.clipboard.writeText(text);
    return true;
  } catch (err) {
    console.error('Error al copiar:', err);
    return false;
  }
}

/**
 * Detecta si está en mobile
 *
 * @returns true si es mobile
 */
export function isMobile(): boolean {
```

```
  return /Android|webOS|iPhone|iPad|iPod|BlackBerry|IEMobile|Opera Mini/i
    navigator.userAgent
  );
}

// ============================================
// 7. UTILIDADES DE URL
// ============================================

/**
 * Obtiene parámetros de URL
 *
 * @returns Objeto con parámetros
 */
export function getUrlParams<T extends Record<string, string>>(): T {
  const params = new URLSearchParams(window.location.search);
  const result: Record<string, string> = {};

  params.forEach((value, key) => {
    result[key] = value;
  });

  return result as T;
}

/**
 * Construye URL con query params
 *
 * @param base - URL base
 * @param params - Objeto con parámetros
 * @returns URL completa
 *
 * @example
 * buildUrl('/api/events', { status: 'active', limit: '10' })
 * → '/api/events?status=active&limit=10'
 */
export function buildUrl(base: string, params: Record<string, string | nu
  const url = new URL(base, window.location.origin);
  Object.entries(params).forEach(([key, value]) => {
    url.searchParams.append(key, String(value));
  });
  return url.toString();
}
```

```typescript
// ============================================
// 8. UTILIDADES ASÍNCRONAS
// ============================================

/**
 * Espera un tiempo determinado (delay)
 *
 * @param ms - Milisegundos a esperar
 * @returns Promise que se resuelve después del delay
 *
 * @example
 * await sleep(1000); // Espera 1 segundo
 */
export function sleep(ms: number): Promise<void> {
  return new Promise(resolve => setTimeout(resolve, ms));
}

/**
 * Debounce una función
 * Útil para búsquedas, resize events, etc.
 *
 * @param func - Función a ejecutar
 * @param wait - Milisegundos a esperar
 * @returns Función debounced
 *
 * @example
 * const debouncedSearch = debounce(searchFunction, 300);
 */
export function debounce<T extends (...args: any[]) => any>(
  func: T,
  wait: number
): (...args: Parameters<T>) => void {
  let timeout: NodeJS.Timeout;

  return function executedFunction(...args: Parameters<T>) {
    const later = () => {
      clearTimeout(timeout);
      func(...args);
    };

    clearTimeout(timeout);
    timeout = setTimeout(later, wait);
  };
}
```

```typescript
/**
 * Throttle una función
 * Limita la frecuencia de ejecución
 *
 * @param func - Función a ejecutar
 * @param limit - Milisegundos mínimos entre ejecuciones
 * @returns Función throttled
 */
export function throttle<T extends (...args: any[]) => any>(
  func: T,
  limit: number
): (...args: Parameters<T>) => void {
  let inThrottle: boolean;

  return function executedFunction(...args: Parameters<T>) {
    if (!inThrottle) {
      func(...args);
      inThrottle = true;
      setTimeout(() => inThrottle = false, limit);
    }
  };
}

// ============================================
// 9. UTILIDADES DE ARRAY
// ============================================

/**
 * Divide un array en chunks
 *
 * @param array - Array a dividir
 * @param size - Tamaño de cada chunk
 * @returns Array de chunks
 *
 * @example
 * chunk([1,2,3,4,5], 2) → [[1,2], [3,4], [5]]
 */
export function chunk<T>(array: T[], size: number): T[][] {
  const chunks: T[][] = [];
  for (let i = 0; i < array.length; i += size) {
    chunks.push(array.slice(i, i + size));
  }
  return chunks;
```

```
}

/**
 * Elimina duplicados de un array
 *
 * @param array - Array con posibles duplicados
 * @returns Array sin duplicados
 */
export function unique<T>(array: T[]): T[] {
  return [...new Set(array)];
}

/**
 * Randomiza un array (Fisher-Yates shuffle)
 *
 * @param array - Array a randomizar
 * @returns Array randomizado
 */
export function shuffle<T>(array: T[]): T[] {
  const newArray = [...array];
  for (let i = newArray.length - 1; i > 0; i--) {
    const j = Math.floor(Math.random() * (i + 1));
    [newArray[i], newArray[j]] = [newArray[j], newArray[i]];
  }
  return newArray;
}

// ============================================
// 10. UTILIDADES DE NÚMERO
// ============================================

/**
 * Genera un número random en un rango
 *
 * @param min - Valor mínimo (inclusive)
 * @param max - Valor máximo (inclusive)
 * @returns Número random
 */
export function randomInt(min: number, max: number): number {
  return Math.floor(Math.random() * (max - min + 1)) + min;
}

/**
 * Clamp un número entre min y max
```

```
 *
 * @param value - Valor a clampear
 * @param min - Valor mínimo
 * @param max - Valor máximo
 * @returns Valor clampeado
 */
export function clamp(value: number, min: number, max: number): number {
  return Math.min(Math.max(value, min), max);
}

// ============================================
// 11. UTILIDADES DE ID
// ============================================

/**
 * Genera un ID único simple
 * Para producción, usar nanoid o UUID
 *
 * @returns ID único
 */
export function generateId(): string {
  return `${Date.now()}-${Math.random().toString(36).substr(2, 9)}`;
}

/**
 * FIN DE LA PARTE 4
 * Continúa en: 06E-CONSTANTES-Y-CONFIG-Parte5.md
 */
```

---

# ✅ VERIFICACIÓN - PARTE 4

- ☐ Función `cn()` para merge de clases Tailwind
- ☐ Utilidades de texto funcionando
- ☐ Validaciones básicas implementadas
- ☐ Funciones de formateo correctas
- ☐ Utilidades de fecha operativas
- ☐ Helpers del DOM configurados
- ☐ Debounce y throttle implementados

---

**Próximo archivo:** Parte 5 - Validators y Formatters