


# ESTÁNDARES DE CODIFICACIÓN OBLIGATORIOS PARA AGENTES

---

 **LECTURA OBLIGATORIA:** Todos los agentes **DEBEN** leer este archivo antes de crear o modificar código.

---

## CONVENCIONES DE NOMBRES UNIFICADAS

---

### ESTRUCTURA DE ARCHIVOS:

flashcard-app-final.js	← Archivo principal (NO tocar estructura)
services/	← Servicios modulares
utils/	← Utilidades compartidas
tests/	← Tests organizados

### NOMENCLATURA DE FUNCIONES:

#### Prefijos Obligatorios:

- `validate` + Objeto → `validateUser()`, `validateForm()`, `validateData()`
- `create` + Objeto → `createDeck()`, `createCard()`, `createUser()`
- `update` + Objeto → `updateDeck()`, `updateCard()`, `updateUser()`
- `delete` + Objeto → `deleteDeck()`, `deleteCard()`, `deleteUser()`
- `get` + Objeto → `getUser()`, `getDeck()`, `getCard()`
- `handle` + Evento → `handleClick()`, `handleSubmit()`, `handleError()`

#### Sufijos para Contexto:

- `...ById()` → `getUserById()`, `getDeckById()`
- `...List()` → `getUserList()`, `getDeckList()`

- `...Count()` → `getUserCount()`, `getDeckCount()`
- `...Exists()` → `userExists()`, `deckExists()`

## NOMENCLATURA DE VARIABLES:

### Constantes (MAYÚSCULAS):

```
const API_BASE_URL = 'https://api.example.com';
const MAX_CARDS_PER_DECK = 100;
const DEFAULT_STUDY_INTERVAL = 24;
```

### Variables (camelCase):

```
const currentUser = getCurrentUser();
const deckList = getDeckList();
const studySession = createStudySession();
```

### Elementos DOM (prefijo 'el'):

```
const elLoginForm = document.getElementById('loginForm');
const elDeckContainer = document.querySelector('.deck-container');
const elSubmitButton = document.querySelector('#submitBtn');
```

---

## GESTIÓN DE DEPENDENCIAS UNIFICADA

---

### IMPORTS/REQUIRES ESTÁNDAR:

#### Orden de Imports:

```
// 1. Librerías externas primero
const axios = require('axios');
const moment = require('moment');

// 2. Servicios internos
const { ApiService } = require('./services/ApiService');
const { AuthService } = require('./services/AuthService');

// 3. Utilidades
const { Utils } = require('./utils/Utils');
const { Validators } = require('./utils/Validators');
```

## Exports Consistentes:

```
// Al final del archivo
module.exports = {
  validateUser,
  createUser,
  updateUser,
  deleteUser
};

// O para clases
module.exports = { UserService };
```

## REGISTRO DE DEPENDENCIAS:

### Antes de crear función, verificar:

1. ¿Existe función similar? → Usar sistema inteligente
2. ¿Necesita dependencias? → Verificar disponibilidad
3. ¿Afecta otras funciones? → Verificar impacto

## Dependencias Comunes Disponibles:

```
// Siempre disponibles en flashcard-app-final.js
- ApiService           ← Para llamadas API
- AuthService          ← Para autenticación
- DeckService          ← Para gestión de mazos
- FlashcardService     ← Para gestión de tarjetas
- StudyingFlash        ← Para algoritmo de estudio
- UIController         ← Para interfaz de usuario
- Utils                ← Utilidades generales
```

---

# SINTAXIS Y ESTILO UNIFICADO

---

## FORMATO DE CÓDIGO:

Indentación: 4 espacios (NO tabs)

Llaves: Estilo JavaScript estándar

```
function validateUser(userData) {  
  if (!userData) {  
    return { valid: false, error: 'Datos requeridos' };  
  }  
  
  if (userData.username.length < 3) {  
    return { valid: false, error: 'Usuario muy corto' };  
  }  
  
  return { valid: true };  
}
```

Comentarios Obligatorios:

```
/**  
 * Valida datos de usuario antes del registro  
 * @param {Object} userData - Datos del usuario a validar  
 * @param {string} userData.username - Nombre de usuario  
 * @param {string} userData.email - Email del usuario  
 * @returns {Object} Resultado de validación {valid: boolean, error?: string}  
 */  
function validateUser(userData) {  
  // Implementación...  
}
```

## MANEJO DE ERRORES ESTÁNDAR:

Try-Catch Obligatorio para APIs:

```
async function createUser(userData) {  
  try {  
    const result = await ApiService.post('/users', userData);  
    Utils.log('✅ Usuario creado exitosamente');  
    return { success: true, data: result };  
  } catch (error) {  
    Utils.log(`❌ Error creando usuario: ${error.message}`, 'error');  
    return { success: false, error: error.message };  
  }  
}
```

## Validación de Parámetros:

```
function processData(data, options = {}) {  
  // Validación obligatoria  
  if (!data) {  
    throw new Error('Parámetro data es requerido');  
  }  
  
  // Valores por defecto  
  const config = {  
    timeout: 5000,  
    retries: 3,  
    ...options  
  };  
  
  // Implementación...  
}
```



## INTEGRACIÓN CON SISTEMAS EXISTENTES



### ANTES DE CREAR CÓDIGO:

#### 1. Verificación Obligatoria:

```
// Usar sistema inteligente SIEMPRE  
const analysis = await coordinator.analyzeAndManageFunction(  
  "descripción de la función",  
  null,  
  targetFile  
);
```

#### 2. Verificar Dependencias:

```
// Verificar que servicios existen  
const requiredServices = ['ApiService', 'AuthService'];  
const available = coordinator.checkServicesAvailable(requiredServices);  
if (!available.allFound) {  
  throw new Error(`Servicios faltantes: ${available.missing.join(', ')}`);  
}
```

### 3. Verificar Impacto:

```
// Verificar funciones que podrían verse afectadas
const impact = coordinator.analyzeImpact(functionName, targetFile);
if (impact.highRisk) {
  Utils.log('⚠ Función de alto impacto - revisar dependencias');
}
```

## DESPUÉS DE CREAR CÓDIGO:

### 1. Limpieza Automática:

```
// Siempre ejecutar después de cambios
await coordinator.executeAutoCleanup();
```

### 2. Verificación de Sintaxis:

```
// Verificar que el código sigue estándares
const syntaxCheck = coordinator.verifyCodingStandards(targetFile);
if (!syntaxCheck.passed) {
  Utils.log('❌ Código no cumple estándares', 'error');
}
```



## COMUNICACIÓN ENTRE AGENTES

---



### EVENTOS ESTÁNDAR:

### Notificaciones Obligatorias:

```
// Al iniciar trabajo
coordinator.notifyAgentStart('AGENT-2', 'Modificando validación de usuarios');

// Al completar trabajo
coordinator.notifyAgentComplete('AGENT-2', 'Validación actualizada exitosamente');

// En caso de error
coordinator.notifyAgentError('AGENT-2', 'Error en validación: datos inválidos');
```

## Verificación de Conflictos:

```
// Antes de modificar archivo
const conflicts = coordinator.checkFileConflicts(targetFile);
if (conflicts.length > 0) {
  Utils.log('⚠️ Conflictos detectados - esperar otros agentes');
  await coordinator.waitForAgents(conflicts);
}
```



## LOGS UNIFICADOS:

### Formato Estándar:

```
// Usar Utils.log SIEMPRE
Utils.log('✅ Función creada exitosamente', 'success');
Utils.log('⚠️ Advertencia: función similar existe', 'warn');
Utils.log('❌ Error: validación falló', 'error');
Utils.log('ℹ️ Información: procesando datos', 'info');
```

---

## ⚠️ REGLAS CRÍTICAS - NUNCA VIOLAR



### PROHIBIDO ABSOLUTAMENTE:

1. NO crear funciones sin verificar duplicados
2. NO modificar flashcard-app-final.js sin análisis previo
3. NO usar nombres de variables genéricos (data, item, temp)
4. NO crear archivos sin verificar estructura existente
5. NO hacer commits sin ejecutar limpieza automática



### OBLIGATORIO SIEMPRE:

1. SÍ usar sistema inteligente antes de crear funciones
  2. SÍ verificar dependencias antes de usar servicios
  3. SÍ seguir nomenclatura estándar
  4. SÍ documentar funciones con JSDoc
  5. SÍ notificar a otros agentes sobre cambios
-

# VERIFICACIÓN FINAL

---


## CHECKLIST ANTES DE COMMIT:


- ☐ ¿Usé sistema inteligente para verificar duplicados?
- ☐ ¿Seguí nomenclatura estándar?
- ☐ ¿Verifiqué dependencias disponibles?
- ☐ ¿Documenté la función con JSDoc?
- ☐ ¿Ejecuté limpieza automática?
- ☐ ¿Notifiqué a otros agentes?
- ☐ ¿El código pasa verificación de sintaxis?

## COMANDO DE VERIFICACIÓN:

```
# Verificar que todo cumple estándares
node scripts/enhanced_agent1_coordinator_fixed.cjs verifyStandards
```

---

 **RECORDATORIO:** Este archivo es la BIBLIA de codificación. Cualquier código que no siga estos estándares será rechazado automáticamente por el sistema de verificación.

 **OBJETIVO:** Código unificado, sin duplicados, con sintaxis consistente y dependencias claras.