

MÓDULO 7

Integraciones: MCP y herramientas externas

Model Context Protocol · Servidores · Herramientas · Diseño

Plan de Formación — OpenAI Codex para Desarrolladores

Nivel: Intermedio-Avanzado · Duración estimada: 3–4 horas

Versión 1.0 — Febrero 2026

Índice

Índice.....	2
1. Introducción.....	4
2. Objetivos de aprendizaje	4
3. Contenidos teóricos	5
3.1 Qué es MCP	5
3.1.1 Tipos de transporte	5
3.2 Configuración de servidores MCP	5
3.2.1 Método 1: codex mcp (CLI)	5
3.2.2 Método 2: Editar config.toml.....	5
3.2.3 Opciones avanzadas de configuración	6
3.2.4 Verificar servidores en el TUI	6
3.3 Servidores MCP populares	7
3.4 Scoping: user vs. proyecto.....	7
3.5 Principios de diseño de herramientas MCP	7
3.5.1 Idempotencia	7
3.5.2 Respuestas estructuradas.....	7
3.5.3 Límites de alcance y permisos	7
3.5.4 Contratos claros.....	7
3.6 Codex como servidor MCP	8
3.7 Seguridad MCP	8
4. Buenas prácticas	9
4.1 Servidores por entorno y rol.....	9
4.2 Sanitizar entradas y salidas	9
4.3 Observabilidad	9
4.4 Añadir instrucciones en AGENTS.md.....	9
5. Errores comunes y cómo evitarlos	10
5.1 Herramientas con efectos colaterales no controlados.....	10
5.2 Contratos ambiguos	10
5.3 Error de sintaxis TOML	10
5.4 Tabla de errores adicionales.....	10
6. Casos de uso reales	11
6.1 Consultar documentación actualizada (OpenAI Docs MCP).....	11
6.2 GitHub integrado (PRs, issues, repos)	11
6.3 Observabilidad: logs y métricas	11
7. Laboratorio práctico (L7) — Conectar un MCP server.....	12
7.1 Prerrequisitos	12

7.2 Paso 1 — Crear proyecto.....	12
7.3 Paso 2 — Configurar MCP con codex mcp add	12
7.4 Paso 3 — Añadir instrucciones en AGENTS.md.....	12
7.5 Paso 4 — Probar MCP en el TUI.....	13
7.6 Paso 5 — Configurar MCP vía config.toml (alternativa).....	13
7.7 Resultado esperado	14
7.8 Limpieza (OBLIGATORIA).....	14
8. Resumen y siguientes pasos	15
8.1 Conceptos clave.....	15
8.2 Preparación para el Módulo 8	15
9. Referencias y recursos	15

1. Introducción

Hasta ahora Codex ha trabajado con el filesystem local: lee ficheros, escribe código, ejecuta tests. Pero los equipos reales necesitan más: consultar tickets de Jira, leer logs de Sentry, acceder a documentación actualizada, interactuar con GitHub PRs o incluso consultar bases de datos.

El Model Context Protocol (MCP) es el mecanismo estándar para conectar Codex a herramientas externas. MCP funciona como un “USB-C para IA”: un protocolo unificado que permite al agente descubrir y usar herramientas de terceros de forma estandarizada.

En este módulo aprenderemos a configurar servidores MCP, diseñar herramientas con contratos claros y conectar Codex al ecosistema de desarrollo del equipo.

2. Objetivos de aprendizaje

#	Objetivo	Evidencia de logro
O1	Configurar servidores MCP en config.toml (stdio y HTTP).	Servidor MCP añadido y verificado con /mcp en TUI.
O2	Usar codex mcp para gestionar servidores desde CLI.	codex mcp add, list, remove ejecutados correctamente.
O3	Entender principios de diseño de herramientas MCP.	Identifica problemas en 3 herramientas mal diseñadas.
O4	Conectar Codex a un MCP server real (OpenAI Docs).	Codex consulta documentación via MCP y responde correctamente.

3. Contenidos teóricos

3.1 Qué es MCP

Model Context Protocol (MCP) conecta modelos a herramientas y contexto externo. Permite a Codex descubrir, invocar y recibir resultados de servicios de terceros sin integraciones ad-hoc. Codex soporta MCP tanto en CLI como en IDE, compartiendo la misma configuración.

3.1.1 Tipos de transporte

Tipo	Descripción	Ejemplo
STDIO	Servidor local ejecutado como subproceso. Codex lo lanza automáticamente.	npx -y @upstash/context7-mcp
Streamable HTTP	Servidor remoto accesible por URL.	https://developers.openai.com/mcp
[i] Configuración compartida La configuración MCP en <code>config.toml</code> es compartida entre CLI e IDE . Configura una vez, usa en ambos clientes. Un error de sintaxis TOML afecta a ambos.		

3.2 Configuración de servidores MCP

3.2.1 Método 1: codex mcp (CLI)

```
# Añadir servidor STDIO
codex mcp add context7 -- npx -y @upstash/context7-mcp

# Añadir servidor STDIO con variables de entorno
codex mcp add github \
--env GITHUB_PAT_TOKEN=ghp_xxxx \
-- npx -y @modelcontextprotocol/server-github

# Añadir servidor HTTP
codex mcp add openaiDocs \
--url https://developers.openai.com/mcp

# Listar servidores configurados
codex mcp list
codex mcp list --json

# Ver configuración de un servidor
codex mcp show context7
codex mcp show context7 --json

# Eliminar un servidor
codex mcp remove context7
```

3.2.2 Método 2: Editar config.toml

```
# ~/.codex/config.toml o .codex/config.toml (trusted)
```

```

# Servidor STDIO
[mcp_servers.context7]
command = "npx"
args = ["-y", "@upstash/context7-mcp"]

# Servidor STDIO con env vars
[mcp_servers.github]
command = "npx"
args = ["-y", "@modelcontextprotocol/server-github"]
[mcp_servers.github.env]
GITHUB_PAT_TOKEN = "ghp_xxxxx"

# Servidor HTTP
[mcp_servers.openaiDocs]
url = "https://developers.openai.com/mcp"

```

3.2.3 Opciones avanzadas de configuración

Opción	Tipo	Default	Descripción
command	string (req)	--	Comando que lanza el servidor STDIO.
args	[string]	[]	Argumentos para el comando.
url	string	--	URL del servidor HTTP (alternativo a command).
env	table	{}	Variables de entorno para el subprocesso STDIO.
bearer_token_env_var	string	--	Env var con bearer token para HTTP.
http_headers	table	{}	Headers estáticos para HTTP.
startup_timeout_sec	int	10	Timeout de arranque del servidor.
tool_timeout_sec	int	60	Timeout de ejecución de herramienta.
enabled	bool	true	false para deshabilitar sin borrar.
enabled_tools	[string]	all	Whitelist de tools expuestas.
disabled_tools	[string]	[]	Blacklist (aplicada después de enabled_tools).

3.2.4 Verificar servidores en el TUI

```

# En el TUI de Codex, escribir:
/mcp

# Muestra todos los servidores MCP activos y sus herramientas

```

3.3 Servidores MCP populares

Servidor	Propósito	Transporte	Config
OpenAI Docs	Buscar y leer documentación OpenAI	HTTP	url = "https://developers.openai.com/mcp"
Context7	Documentación actualizada de librerías	STDIO	npx -y @upstash/context7-mcp
GitHub MCP	PRs, issues, repos de GitHub	HTTP	url = "https://api.githubcopilot.com/mcp/"
Filesystem	Acceso controlado al filesystem	STDIO	npx -y @modelcontextprotocol/server-filesystem /path
Playwright	Automatización de browser, tests E2E	STDIO	npx -y @playwright/mcp@latest

Catálogo: Explora más servidores MCP en la documentación oficial y en la comunidad.

3.4 Scoping: user vs. proyecto

Scope	Ubicación	Quién lo ve	Caso de uso
User (global)	~/.codex/config.toml	Solo tú, en todos los repos	Servidores personales: docs, GitHub
Proyecto	./codex/config.toml	Todo el equipo (si trusted)	Servidores específicos del proyecto

Importante: Los servidores MCP de proyecto solo se cargan si el proyecto está marcado como **trusted**. Esto evita que repos maliciosos configuren servidores MCP no autorizados.

3.5 Principios de diseño de herramientas MCP

Al crear o elegir servidores MCP, aplica estos principios para que Codex sea fiable:

3.5.1 Idempotencia

Las herramientas de **lectura** deben ser idempotentes: llamarlas múltiples veces con los mismos parámetros debe devolver el mismo resultado sin efectos secundarios. Las herramientas de **escritura** deben declarar explícitamente sus efectos.

3.5.2 Respuestas estructuradas

- Devolver datos estructurados (JSON), no texto libre.
- Incluir campos de error claros cuando falle.
- Limitar el tamaño de la respuesta (no volcar tablas de 10K filas).

3.5.3 Límites de alcance y permisos

- Principio de mínimo privilegio: exponer solo las tools necesarias.
- Usar enabled_tools / disabled_tools para restringir.
- No exponer herramientas destructivas (DELETE, DROP) sin approval.

3.5.4 Contratos claros

- Nombre descriptivo de cada tool (list_open_tickets, not do_stuff).

- Descripción precisa de qué hace y qué parámetros acepta.
- Documentar side effects explícitamente.

⚠ Sanitización de secretos

Nunca incluir tokens, passwords o API keys en las respuestas de herramientas MCP. Los datos de herramientas se inyectan en el contexto del modelo y podrían filtrarse en la salida.

3.6 Codex como servidor MCP

Codex CLI puede ejecutarse como servidor MCP para que otros agentes lo consuman:

```
# Iniciar Codex como MCP server
npx -y codex mcp-server

# Expone dos herramientas:
# - codex(): iniciar conversación
# - codex-reply(): continuar conversación

# Útil para multi-agent workflows con OpenAI Agents SDK
```

3.7 Seguridad MCP

- Los servidores MCP de proyecto requieren trusted project.
- Enterprise: requirements.toml define allowlist de servidores MCP permitidos.
- Codex puede solicitar aprobación para tool calls que declaran side effects.
- Si [mcp_servers] está presente pero vacío, Codex deshabilita todos los MCP.
- required = true en un servidor: si falla al inicializar, codex exec aborta.

4. Buenas prácticas

4.1 Servidores por entorno y rol

- Dev: GitHub MCP con permisos de lectura + Context7 para docs.
- Staging: herramientas de observabilidad (logs, métricas) en read-only.
- Producción: ningún MCP o solo herramientas de lectura estrictamente necesarias.

4.2 Sanitizar entradas y salidas

- No filtrar secretos, tokens ni datos PII en respuestas de herramientas.
- Truncar respuestas grandes (usar paginación o límites).
- Validar parámetros de entrada antes de ejecutar la acción.

4.3 Observabilidad

- Habilitar OTel: codex.tool_result registra cada invocación MCP.
- Correlacionar por sesión: las transcripciones en `~/.codex/sessions/` incluyen tool calls.
- Usar `/mcp` en el TUI para verificar servidores activos antes de cada sesión.

4.4 Añadir instrucciones en AGENTS.md

Para que Codex use servidores MCP automáticamente, añade instrucciones en AGENTS.md:

```
# En AGENTS.md del proyecto:  
Always use the OpenAI developer documentation MCP server  
if you need to work with the OpenAI API, Codex, or Apps SDK  
without me having to explicitly ask.
```

5. Errores comunes y cómo evitarlos

5.1 Herramientas con efectos colaterales no controlados

Problema: Un servidor MCP que expone DELETE o UPDATE sin restricciones. Codex podría invocar una herramienta destructiva si interpreta el prompt como una instrucción para “limpiar” datos.

Solución: Separar herramientas de lectura y escritura. Usar disabled_tools para bloquear las destructivas. Las herramientas de escritura deben declarar side effects para que Codex pida aprobación.

5.2 Contratos ambiguos

Problema: Tool llamada “do_stuff” sin descripción clara. Codex no puede decidir cuándo usarla correctamente.

Solución: Nombres descriptivos (list_open_tickets, create_github_issue). Descripciones que expliquen parámetros, retorno y side effects.

5.3 Error de sintaxis TOML

Problema: Un error de TOML en config.toml rompe tanto CLI como IDE simultáneamente (configuración compartida).

Solución: Validar TOML antes de guardar. Probar con codex mcp list después de cada cambio.

5.4 Tabla de errores adicionales

Error	Consecuencia	Prevención
Timeout de arranque <10s para servidor pesado	MCP falla silenciosamente	Ajustar startup_timeout_sec
No configurar trusted project	MCP de proyecto ignorados	Marcar en [[project_trust]]
Secretos en respuestas de herramientas	Tokens filtrados en output del modelo	Sanitizar respuestas del servidor
Servidor MCP con acceso total al filesystem	Riesgo de ex filtración	Usar enabled_tools para restringir

6. Casos de uso reales

6.1 Consultar documentación actualizada (OpenAI Docs MCP)

Escenario: Equipo usando la API de OpenAI. Necesitan que Codex consulte la documentación oficial automáticamente en lugar de depender de conocimiento potencialmente desactualizado.

```
# config.toml
[mcp_servers.openaiDocs]
url = "https://developers.openai.com/mcp"

# AGENTS.md
Always use the OpenAI developer documentation MCP server
for OpenAI API questions.
```

6.2 GitHub integrado (PRs, issues, repos)

Escenario: Codex lee PRs abiertas, crea issues y revisa código directamente desde GitHub.

```
[mcp_servers.github]
url = "https://api.githubcopilot.com/mcp/"
bearer_token_env_var = "GITHUB_PAT_TOKEN"
```

6.3 Observabilidad: logs y métricas

Escenario: Equipo SRE usa MCP para que Codex lea logs de aplicación y sugiera diagnósticos.

Se configura un servidor MCP personalizado que expone herramientas read-only: `search_logs`, `get_metrics`, `list_alerts`. Las herramientas destructivas (`acknowledge_alert`) se configuran como `disabled_tools`.

7. Laboratorio práctico (L7) — Conectar un MCP server

⌚ Objetivo del laboratorio

Configurar el servidor MCP **OpenAI Docs** (real, gratuito, sin autenticación) y verificar que Codex lo usa para responder preguntas sobre la API de OpenAI.

7.1 Prerrequisitos

- Codex CLI instalado y autenticado.
- Git instalado.

7.2 Paso 1 — Crear proyecto

Tiempo estimado: 3 minutos

```
mkdir /tmp/codex-lab07 && cd /tmp/codex-lab07
git init

cat > .gitignore << 'EOF'
__pycache__/
*.pyc
EOF

git add -A && git commit -m "chore: scaffold lab07"
```

7.3 Paso 2 — Configurar MCP con codex mcp add

Tiempo estimado: 5 minutos

```
# Añadir OpenAI Docs MCP (HTTP, gratuito)
codex mcp add openaiDocs \
--url https://developers.openai.com/mcp

# Verificar
codex mcp list
# Esperado: openaiDocs listado con URL correcta

codex mcp show openaiDocs
# Muestra la configuración completa
```

7.4 Paso 3 — Añadir instrucciones en AGENTS.md

Tiempo estimado: 2 minutos

```
cat > AGENTS.md << 'EOF'
# Project Instructions

Always use the OpenAI developer documentation MCP server
```

```
if you need to work with the OpenAI API, Codex, or Apps SDK
without me having to explicitly ask.
```

```
EOF
```

```
git add AGENTS.md && git commit -m "docs: AGENTS.md con MCP"
```

7.5 Paso 4 — Probar MCP en el TUI

Tiempo estimado: 10 minutos

```
cd /tmp/codex-lab07
codex --full-auto
```

TEST 1: Verificar servidores activos

```
/mcp
# Esperado: openaiDocs listado con sus herramientas
```

TEST 2: Preguntar sobre la API de OpenAI

```
# Prompt en el TUI:
"¿Cuáles son los campos requeridos del Responses API de OpenAI?"

# Observación: Codex debe usar el MCP server para
# buscar en la documentación oficial y dar una respuesta
# actualizada, NO basarse solo en su conocimiento interno.
```

TEST 3: Pedir código basado en documentación actual

```
# Prompt en el TUI:
"Crea un ejemplo de uso de la API Responses de OpenAI
en Python. Consulta la documentación para usar los
parámetros correctos y actualizados."

# Observación: Codex consulta el MCP, obtiene el schema
# actual y genera código coherente con la documentación.
```

7.6 Paso 5 — Configurar MCP vía config.toml (alternativa)

Tiempo estimado: 5 minutos

```
# Crear config de proyecto con MCP
mkdir -p .codex

cat > .codex/config.toml << 'EOF'
[mcp_servers.openaiDocs]
url = "https://developers.openai.com/mcp"

# También se puede añadir un servidor STDIO de ejemplo:
# [mcp_servers.filesystem]
```

```
# command = "npx"
# args = ["-y", "@modelcontextprotocol/server-filesystem", "/tmp"]
EOF

git add .codex/ && git commit -m "config: MCP en config.toml"
```

7.7 Resultado esperado

Verificación	Resultado esperado
codex mcp list	openaiDocs listado
/mcp en TUI	Herramientas del servidor visibles
Pregunta sobre API OpenAI	Codex consulta MCP antes de responder
Código generado	Usa parámetros actualizados de la documentación

7.8 Limpieza (OBLIGATORIA)

⚠️ Limpieza de recursos

Ejecutar al finalizar:

```
# Eliminar MCP server de config global (si se añadió ahí)
codex mcp remove openaiDocs

# Borrar proyecto
rm -rf /tmp/codex-lab07

ls /tmp/codex-lab07 2>/dev/null \
&& echo 'ERROR' || echo 'OK: limpio'
```

8. Resumen y siguientes pasos

8.1 Conceptos clave

Concepto	Detalle
MCP	Model Context Protocol: estándar para conectar modelos a herramientas externas
Transportes	STDIO (subproceso local) o Streamable HTTP (servidor remoto)
codex mcp	Subcomando CLI para add/list/show/remove servidores
config.toml	[mcp_servers.nombre] con command/args (STDIO) o url (HTTP)
Opciones	enabled_tools, disabled_tools, timeouts, env, bearer_token_env_var
/mcp en TUI	Verificar servidores activos y herramientas disponibles
Trusted projects	MCP de proyecto solo se carga si está trusted
Diseño de herramientas	Idempotencia, respuestas estructuradas, contratos claros
Codex como MCP server	npx codex mcp-server para multi-agent workflows
Seguridad	Sanitizar secretos, restringir tools, requirements.toml enterprise

8.2 Preparación para el Módulo 8

En el Módulo 8 abordaremos Codex Cloud y GitHub Action: cómo delegar tareas a entornos remotos, ejecutar Codex en CI/CD y escalar el uso del agente. Para prepararse:

- Revisar: <https://developers.openai.com/codex/cloud/>
- Revisar: <https://developers.openai.com/codex/noninteractive/>
- Revisar: <https://developers.openai.com/codex/github-action/>

9. Referencias y recursos

- MCP overview: <https://developers.openai.com/codex/mcp/>
- CLI reference (mcp subcommand): <https://developers.openai.com/codex/cli/reference/>
- CLI features (MCP): <https://developers.openai.com/codex/cli/features/>
- Config reference: <https://developers.openai.com/codex/config-reference/>
- Config sample: <https://developers.openai.com/codex/config-sample/>
- OpenAI Docs MCP: <https://developers.openai.com/resources/docs-mcp/>
- Codex as MCP server: <https://developers.openai.com/codex/guides/agents-sdk/>
- Security (MCP enterprise): <https://developers.openai.com/codex/security/>
- GitHub MCP server: github.com/github/github-mcp-server