Este es el contenido de mis archivos AGENTS.md y CLAUDE.md para Codex, Claude Code y OpenCode. Este contenido debe estar dentro de los archivos **~/.claude/CLAUDE.md**, **~/.codex/AGENTS.md** y **~/.config/opencode/AGENTS.md**:

```
## Código

Para todo mi código usa las siguientes reglas que son OBLIGATORIAS y tienen prioridad sobre cualquier otra
instrucción:

### Indentación

 - Usa siempre una indentación de 2 espacios.
 - Debes reemplazar cualquier tabulación por exactamente 2 espacios.

### Estructura

 - No simplifiques la lógica al traducir código entre lenguajes.
 - No simplifiques la lógica al modificar código escrito previamente por mí.
 - No reinterpretes ni "optimices" mi forma de programar.
 - Respeta estrictamente mi estructura original.

### Convenciones de nombres

Se deben seguir estas convenciones de nombres estilo CamelCase precedidos por iniciales en minúscula, de esta
forma:

 - **Variables**: Precedido por `v` minúscula (ej: `vContador`).
 - **Constantes**: Precedido por `c` minúscula (ej: `cMaximo`).
 - **Arrays**: Precedido por `a` minúscula (ej: `aUsuarios`).
 - **Listas**: Precedido por `l` minúscula (ej: `lUsuarios`).
 - **Diccionarios**: Precedido por `d` minúscula (ej: `dLibrosSciFi`).
 - **Métodos**: Precedido por `m` minúscula (ej: `mEspecial`).
 - **Lista de diccionarios**: Precedido por `ld` minúscula (ej: `ldEstoEsUnaListaDeDiccionarios`).
 - **Funciones**: Precedido por `f` minúscula (ej: `fProcesarDatos`).
 - **Parámetros**: Cuando se define una función, el nombre de los parámetros debe empezar por `p` (ej: `pCiudad`)

#### Ejemplo

```python
def fCalcularTotal(pCantItems, pPrecioUnitario):
  cImpuesto = 0.21
  vSubtotal = pCantItems * pPrecioUnitario
  vTotal = vSubtotal * (1 + cImpuesto)
  return vTotal
aResultados = [1, 2, 3]
dConfiguracion = {"clave": "valor"}
```

### Bash

 - Shebang obligatorio y exacto: `#!/bin/bash`
 - Siempre deja una línea en blanco después del shebang.
 - No usar `awk`, a menos que sea estrictamente necesario. Siempre dale preferencia al uso de `sed`.
 - Prohibido usar EOF o here-documents en cualquier contexto.
 - Los scripts deben ser reproducibles y no interactivos cuando sea posible.
 - Siempre prefiere `< /dev/tty` antes que stdin, así el script nunca avanza sin la interacción real del usuario.

#### Control de errores en bash

 - Debes usar siempre `set -euo pipefail`.
 - Debes definir una función `fCleanup` y registrar `trap fCleanup EXIT`.
 - Debes encapsular la lógica principal en una función `fMain`.
 - Debes ejecutar `fMain` con `if ! fMain; then ... fi`.
 - No debes usar bloques `{ ... } || { ... }` como mecanismo principal de control de errores.

#### Ejemplo de plantilla

```bash
#!/bin/bash
```
```

```
set -euo pipefail

fCleanup() {
  # limpieza
  :
}

trap fCleanup EXIT

fMain() {
  # lógica principal
  :
}

if ! fMain; then
  echo "error"
fi
```

### Python

 - Pon un shebang siempre que el script se pueda ejecutar directamente.
 - El código del shebang debe ser siempre `#!/usr/bin/env -S PYTHONDONTWRITEBYTECODE=1 python3`.
 - Deja una línea en blanco después del shebang.
 - Si un script escucha en un puerto, cerrar previamente cualquier socket abierto en ese mismo puerto.
 - Instalar paquetes con: `python3 -m pip install NombreDelPaquete --break-system-packages`.

#### Control de errores en Python

 - Usa try/except/finally siempre que se pueda.

#### Namespace packages implícitos

 - Crea código únicamente compatible con python 3.13 o superior, porque Python 3.13 soporta namespace packages implícitos. Eso lo quiero así porque no me gusta que entre los archivos del proyecto hayan archivos `__init__.py`

### PHP

#### Control de errores en PHP

 - Usa try/catch/finally siempre que se pueda.

### Apache

Cuando crees webs con Apache, usa estos templates de `.conf`:

#### VirtualHost HTTP (puerto 80)

```apache
<VirtualHost *:80>
  Redirect permanent / https://dominio.com/
  ServerName dominio.com
  ServerAlias www.dominio.com
  DocumentRoot /var/www/dominio.com
  <Directory "/var/www/dominio.com">
    Require all granted
    Options FollowSymLinks
    AllowOverride All
  </Directory>
  ServerAdmin admin@dominio.com
  ErrorLog  /var/www/dominio.com-logs/error.log
  CustomLog /var/www/dominio.com-logs/access.log combined
  <Directory "/var/www/dominio.com/__">
    AuthType Basic
    AuthName "Restricted Content"
    AuthUserFile /etc/apache2/.htpasswd
```

```
    Require valid-user
  </Directory>
  RewriteEngine on
  RewriteCond %{SERVER_NAME} =www.dominio.com [OR]
  RewriteCond %{SERVER_NAME} =dominio.com
  RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]
</VirtualHost>
```

#### VirtualHost HTTPS (puerto 443)

```apache
<IfModule mod_ssl.c>
  <VirtualHost *:443>
    #Redirect permanent / http://dominio.com/
    RemoteIPProxyProtocol On
    ServerName dominio.com
    ServerAlias www.dominio.com
    DocumentRoot /var/www/dominio.com
    <Directory "/var/www/dominio.com">
      Require all granted
      Options FollowSymLinks
      AllowOverride All
    </Directory>
    ServerAdmin admin@dominio.com
    ErrorLog  /var/www/dominio.com-logs/error.log
    CustomLog /var/www/dominio.com-logs/access.log combined
    Include               /etc/letsencrypt/options-ssl-apache.conf
    SSLCertificateFile    /etc/letsencrypt/live/dominio.com/fullchain.pem
    SSLCertificateKeyFile /etc/letsencrypt/live/dominio.com/privkey.pem
  </VirtualHost>
</IfModule>
```

#### Puertos personalizados

Si necesitas escuchar en puertos distintos al 80 y al 443, aplica los siguientes cambios:

**En el bloque HTTP**, sustituye:
```apache
<VirtualHost *:80>
```

por:
```apache
Listen 11080
<VirtualHost *:11080>
```

**En el bloque HTTPS**, sustituye:
```apache
<IfModule mod_ssl.c>
  <VirtualHost *:443>
```

por:
```apache
<IfModule mod_ssl.c>
  Listen 11443
  <VirtualHost *:11443>

```

### Archivos CLAUDE.md y AGENTS.md en la raíz de cada proyecto

Cada proyecto debe tener SIEMPRE en su carpeta raíz, estos dos archivos .md:
 - **CLAUDE.md**: Que dentro tendrá, como mínimo, una línea OBLIGATORIA con el texto `@AGENTS.md`.

```
 - **AGENTS.md**: Que dentro tendrá. como mínimo, estas líneas OBLIGATORIAS:

```
# AGENTS.md

Este archivo proporciona guías y directivas para trabajar con el código y el contenido de este proyecto.

@ReglasPersonales.md
```
Si un proyecto NO TIENE esos dos archivos en su carpeta raíz, créalos y popúlalos con las líneas obligatorias.


### Tildes en las palabras en español

Siempre que comentes código en español, pongas cadenas de texto que serán visibles en la app o populos los archivos .md con palabras en español, asegúrate de que esas palabras lleven tilde donde deben llevarla.
```

Teniendo esos archivos con ese texto podremos hacer que todos nuestros proyectos sigan esas reglas básicas. Luego, en la carpeta raíz de cada proyecto, podemos crear un archivo llamado **ReglasPersonales.md** con las reglas personales que queramos. En mi caso, este es el contenido de mi archivo ReglasPersonales.md:

```
## Archivos .md en la raíz del proyecto

Este proyecto deberá tener los siguientes archivos .md en la carpeta raíz:

 - **README.md**: con las explicaciones típicas en inglés que suelen estar en los README (para que la gente menos técnica entienda que funcionalidad tiene el proyecto, como se instala en Debian y como se ejecuta una vez instalado).
 - **CODE.md***: explicación técnica en inglés del codebase para personas con conocimiento técnico y modelos de lenguaje de inteligencia artificial, pensada para consultarse de forma quirúrgica (saltar directamente a la sección concreta que se necesita) sin tener que leer todo el código ni todo el documento. Su estructura obligatoria se detalla en la sección «Estructura de CODE.md».
 - **MANUAL.md**: será el manual de uso de la aplicación en inglés. Servirá al usuario para que entienda como se usa la aplicación, como se interactúa con las diferentes funcionalidades de la misma y toda la información que creas que será relevante para que una persona que no conoce como se opera con la app, pueda aprender a utilizarla y ponerse en marcha inmediatamente.
 - **README.es-ES.md**: que será la versión en español de España, del archivo README.md.
 - **CODE.es-ES.md**: que será la versión en español de España del archivo CODE.md.
 - **MANUAL.es-ES.md**: que será la versión en español de España del archivo MANUAL.md.
 - **README.es-AR.md**: que será la versión en español de Argentina del archivo README.md.
 - **CODE.es-AR.md**: que será la versión en español de Argentina del archivo CODE.md.
 - **MANUAL.es-AR.md**: que será la versión en español de Argentina del archivo MANUAL.md

Cada vez que se hagan cambios en el código deben actualizarse esos archivos para reflejar lo modificado. Todos los archivos, tanto los README como los CODE como los MANUAL deben estar SIEMPRE actualizados para reflejar el estado actual del código.

### Estructura de CODE.md

CODE.md debe poder consultarse de forma quirúrgica: empezará con un índice y usará encabezados estables que sirvan de ancla, para que una persona o un modelo de lenguaje salte directamente a la sección que necesita sin leer el archivo entero. Como mínimo contendrá:

 1. Arquitectura y decisiones de diseño (en prosa): visión general y el porqué de la estructura. Es lo único que no puede deducirse automáticamente del código, así que es la parte más valiosa.
 2. Mapa de módulos: tabla con `módulo | ruta | responsabilidad | depende de | usado por`.
 3. Índice de símbolos clave: tabla con `símbolo (función/clase/método) | archivo:línea | qué hace`. Solo los públicos o relevantes, no todos.
 4. Flujos principales (call paths): para cada operación importante, la secuencia de llamadas desde el punto de entrada hasta la salida.
 5. Mapa de entradas/rutas (para webs, APIs o CLIs): tabla con `ruta/URL/comando | handler | archivo`.
 6. Análisis de impacto: para cada componente crítico o compartido, qué se ve afectado si se modifica.
 7. Puntos de extensión: cómo añadir un caso típico nuevo (endpoint, comando, etc.).

El índice de símbolos y el mapa de módulos se mantienen a nivel de módulo y de símbolo público o clave, nunca símbolo a símbolo de forma exhaustiva (eso deriva y se desactualiza; es trabajo de una herramienta de indexado, no de un documento). Cada cambio en el código debe actualizar la fila del símbolo, módulo o flujo afectado en CODE.md,
```

````
no solo la prosa.

## Archivo .gitignore

Deberá haber un archivo llamado `.gitignore` en la carpeta raíz del proyecto con, AL MENOS, este texto:

```
# El propio gitignore
.gitignore

# La carpeta .agents
.agents/
# El archivo AGENTS.md
AGENTS.md

# La carpeta .claude
.claude/
# El archivo CLAUDE.md
CLAUDE.md

# Los archivos de la carpeta _
_/*

# Carpetas y archivos temporales
__pycache__/
*.pyc
```
````

.