Escuela Politecnica Nacional

Facultad de Ingeniería de Sistemas

Construcción y Evolución de Software

Versión:	1.0
Grupo:	5
Fecha:	Noviembre 2025

Ambiente de Desarrollo

Este documento describe las herramientas seleccionadas, la configuración mínima y el flujo de trabajo Git usado en el proyecto `Braile_checked_web`. También incluye comandos útiles para poner el entorno en marcha, generar la documentación y las reglas de ramificación que aplicamos (Main, develop, feature, hotfix, documentation).

Herramientas seleccionadas

- **Lenguaje:** Python (3.7+ recomendado). Se probó con Python 3.11.
- **Framework web:** Flask (`Flask==3.0.0`) — backend ligero para servir la UI y la API de conversión a Braille.
- **Dependencias:** ver `requirements.txt` para versiones exactas (Werkzeug, Sphinx, etc.).
- **Control de versiones:** Git (CLI) y hospedaje en **GitHub**.
- **Generador de documentación:** Sphinx (`sphinx==8.x`) con tema `sphinx-rtd-theme`.
- **Editor / IDE recomendado:** Visual Studio Code.
- **Navegador:** Chrome (navegadores modernos con buen soporte Unicode).

Configuración del entorno (local)

1. Clonar el repositorio:

```
git clone https://github.com/Construccion-y-Evolucion-Pry/Braile_checked_web.git
cd Braile_checked_web
```

2. Crear y activar un entorno virtual (Windows CMD):

```
python -m venv .venv
.venv\Scripts\activate.bat
```

3. Instalar dependencias:

```
pip install -r requirements.txt
```

4. Ejecutar la aplicación en desarrollo:

```
python app.py
```

Abrir `http://localhost:5000` en el navegador.

Generar la documentación

Hay scripts incluidos para simplificar la generación de documentación.

- En Windows (CMD):

```
generar_docs.bat
```

- En entornos Unix (Git Bash):

```
./generar_docs.sh
```

Manualmente con Sphinx:

```
python -m sphinx -b html Documentation/source Documentation/build/html
```

Para abrir la documentación generada en Windows como interfaz asimilada de pythondoc:

```
start Documentation\\build\\html\\index.html
```

Flujo de trabajo Git y políticas de ramas

Se aplica el estándar de Git/GitHub con las siguientes ramas principales:

- `Main`: rama de producción; siempre estable. Los releases finalizados se encuentran aquí (etiquetas semánticas opcionales).
- `develop`: rama de integración; aquí se fusionan las `feature/*` completadas para preparar la siguiente versión.
- `feature/*`: ramas de trabajo para nuevas funcionalidades. Nomenclatura: `feature/nombre-descriptivo`.
- `hotfix/*`: correcciones críticas que deben salir inmediatamente en producción. Se crean desde `Main` y, después de aplicar, se mergean a `Main` y `develop`.
- `documentation/*`: ramas específicas para cambios importantes de la documentación (o se pueden usar PRs directos a `documentation` si se prefiere).

Reglas y flujo recomendado:

1. Crear una rama `feature` desde `develop`:

```
git checkout develop
git pull origin develop
git checkout -b feature/nueva-funcionalidad
```

2. Trabajar localmente, commits atómicos y descriptivos:

```
git add .
git commit -m "feat: añadir conversor de signos especiales"
```

3. Mantener la rama actualizada con `develop` (rebase o merge según política del equipo):

```
git fetch origin
git rebase origin/develop
```

Buenas prácticas y comandos útiles

- Escribir mensajes de commit claros y en inglés/español consistente según el equipo.
- Ejecutar pruebas locales antes de abrir PRs (si hubiera tests).
- Evitar commits con credenciales o archivos grandes sin seguimiento (usar `.gitignore`).
- Revisar cambios de `requirements.txt` y actualizarlos con `pip freeze > requirements.txt` cuando se añadan paquetes.
- Para sincronizar la rama local con la remota:

```
git fetch origin
git checkout develop
git pull origin develop
```

Consideraciones de accesibilidad y factor humano

Este proyecto tiene un propósito social: facilitar el acceso a la información a personas con discapacidad visual mediante la conversión de texto a Braille. Al tomar decisiones técnicas se deben priorizar:

- **Legibilidad y compatibilidad Unicode:** probar los símbolos Braille en distintos navegadores y plataformas.
- **Interfaz simple e intuitiva:** minimizar pasos para el usuario final (cortar, pegar, convertir, copiar resultado).
- **Documentación clara:** instrucciones de uso accesibles y ejemplos para educadores o usuarios finales.
- **Privacidad:** no almacenar textos de usuarios en servidores públicos sin consentimiento.

Recordar que el software no es un fin en sí mismo, sino una herramienta para mejorar la calidad de vida: el criterio de aceptación incluye el impacto real en usuarios finales.

Recursos y referencias rápidas

- `requirements.txt` — dependencias del proyecto
- `app.py` — servidor Flask y lógica de conversión
- `templates/` y `static/` — frontend

- Documentación generada: `Documentation/build/html/index.html`, ejecutarlo a partir de `start Documentation/build/html/index.html`