🧩 Projektname:

NetPipeRemoteDotNet – Automatisierter Build-Monitor mit Trigger- und Git-Integration

---

🔍 Projektziel:

Das Projekt dient zur **automatisierten Überwachung, Prüfung und Ausführung von Build-Vorgängen** für ein .NET-Projekt (z. B. eine DLL oder EXE), sobald entweder ein **manueller Trigger** oder eine **Änderung im Git-Repository** erkannt wird. Die Ergebnisse werden standardisiert in eine `result.txt`-Datei geschrieben und per POST an eine Webschnittstelle (z. B. `http://t78.ch/apps/netpipe/result.ashx`) übermittelt.

⚙️ Systemvoraussetzungen:

- Windows mit installiertem `.NET SDK` (mind. .NET 6.0)
- Python 3.x mit den Standardmodulen (`requests`, `subprocess`, etc.)
- Internetzugang für:
  - Git Pull
  - IP-Auflösung (https://api.ipify.org)
  - Remote-Kommunikation (`result.ashx`)
- Schreibrechte im Arbeitsverzeichnis

📁 Projektstruktur (Beispiel):

```
NetPipeRemoteDotNet\
│   netpipe_runner.py
│   netpipe_settings.json
│
├──NetProject\
│   └──bin\
│       └──Release\
│           └──net6.0\
│               ├─ NetProject.dll
│               ├─ NetProject.exe
│               └─ ...
```

🧩 Konfigurationsdatei: `netpipe_settings.json`

```
{
  "git_enabled": true,
  "git_url": "https://github.com/tb-software/netpipe.git",
  "git_branch": "main",
  "project_folder": "G:/Development/TBS/netpipe",
  "solution_file": "NetProject.sln",
  "project_file": "",
  "output_dll": "NetProject/bin/Release/net6.0/NetProject.dll",
  "dotnet_target": "net6.0",
  "netpipe_base_url": "http://t78.ch/apps/netpipe/result.ashx",
  "trigger_filename": "triggers/build_trigger.json",
  "result_filename": "result.txt",
  "interval_seconds": 10
}
```

🔄 Trigger-Mechanismus:

Das Skript überwacht alle 10 Sekunden:

1. **Git Pull**: Holt automatisch Updates vom angegebenen Git-Branch. Erkennt Änderungen über den Pull-Output.
2. **Trigger-Datei**: Prüft, ob im JSON der Key `"trigger": true` gesetzt wurde.
3. Wenn eine der beiden Bedingungen erfüllt ist, startet der Build-Vorgang.

🛠 Build-Prozess:

- Führt den Befehl `dotnet build NetProject.sln -c Release` aus
- Prüft, ob die Zieldatei (`NetProject.dll`) im erwarteten Pfad existiert
- Misst Dateigröße und Exit-Code
- Holt lokale und externe IP-Adresse
- Fügt die Ausgabe von `tree /a /f` über das Ausgabe-Verzeichnis hinzu

---

📝 Beispielausgabe: `result.txt`

Jedes Ergebnis enthält folgende standardisierte Struktur:

✅ Erfolgreicher Build

```
=== Build started: 2025-06-07 09:36:48 ===
🔨 Build command: dotnet build NetProject.sln -c Release
💻 Host: TBE2019, Local IP: 192.168.1.233, External IP: 178.21.2
📄 Exit code: 0
✅ Build OK: NetProject.dll (4608 bytes)

---

  NetProject -> G:\Development\...\NetProject.dll
  Der Buildvorgang wurde erfolgreich ausgeführt.
    0 Warnung(en)
    0 Fehler

Verstrichene Zeit 00:00:01.84

--- Output Folder Structure ---

G:\DEVELOPMENT\TBS\NETPIPE\NETPROJECT\BIN\RELEASE\NET6.0
    NetProject.deps.json
    NetProject.dll
    NetProject.exe
    NetProject.pdb
    NetProject.runtimeconfig.json
```

❌ Fehlerhafter Build

```
=== Build started: 2025-06-07 09:35:12 ===
🔨 Build command: dotnet build NetProject.sln -c Release
💻 Host: TBE2019, Local IP: 192.168.1.233, External IP: 178.21.2
📄 Exit code: 1
✅ Build OK: NetProject.dll (4608 bytes)

---

G:\...\Program.cs(2,6): error CS1001: Bezeichner erwartet.
G:\...\Program.cs(2,6): error CS1002: ; erwartet.
    0 Warnung(en)
    2 Fehler

Verstrichene Zeit 00:00:00.71

--- Output Folder Structure ---

G:\DEVELOPMENT\TBS\NETPIPE\NETPROJECT\BIN\RELEASE\NET6.0
    NetProject.deps.json
    NetProject.dll
    NetProject.exe
    NetProject.pdb
    NetProject.runtimeconfig.json
```

> Hinweis: Auch bei **Exit-Code 1** kann eine `.dll` vorhanden sein – zur Debug-Zwecken wird sie dennoch als „OK" mit Dateigröße aufgeführt.

📤 Serverkommunikation:

Ergebnis wird als `utf-8` Text-Payload via `POST` an die `result.ashx`-Schnittstelle gesendet, mit folgendem Parameter:

```
file=result.txt
```

✅ Ziel & Nutzen:

- Ermöglicht **remote-gesteuertes Build-Feedback**
- Unterstützt Codex oder andere KI-Systeme bei der **Fehlerdiagnose**
- **Build-Trigger manuell oder durch Commit**
- **Alle relevanten Daten** (Zeit, Build-Log, Dateigrößen, Netzwerkinfo, Verzeichnisstruktur) in einer Datei
- Grundlage für KI-gestützte automatische Codekorrekturen