Terminal “GUI” for EasyEngine (Debian) □□□

> Not tested yet.

> □ I've installed EasyEngine on my Debian root server. Could you write me a script that displays a GUI in the terminal where I can do everything? So that it runs the ee commands in the background but displays everything graphically for me. For example, I'd like to be able to view, modify, and create all my websites, etc.

You can absolutely build a *terminal UI (TUI)* that **wraps** `ee`, runs commands “in the background,” and lets you **list / create / update / delete / enable / disable** sites, view info, manage SSL, etc.

The most maintainable approach on Debian is:

- **Python +** `Textual` (modern TUI framework, mouse/keyboard, panels, logs) □
- Spawn `ee` **subprocesses**, stream output to a “log pane,” and show progress/status.

Below is a *working starter app* you can extend into “do everything.” It already supports:

1. **List sites** (and refresh)
2. **Show site info**
3. **Enable / Disable**
4. **Delete** (with confirmation)
5. **Create** (HTML/WP, optional SSL)
6. **Update SSL** (set/renew)

> ⚠ **Security note:** This kind of wrapper typically needs to run as **root** (or a user with permission to run Docker + manage `/opt/easyengine`). Be careful about storing admin passwords (for WP create) and about allowing arbitrary command execution.

---

1) Install prerequisites (Debian)

```
apt-get update
apt-get install -y python3 python3-pip
python3 -m pip install --upgrade pip
python3 -m pip install textual rich
```

Verify EasyEngine works:

```
ee --version
ee site list --format=json
```

---

2) Save the TUI script

Create a file: `ee-tui.py`

```
#!/usr/bin/env python3
import asyncio
import json
import shlex
from dataclasses import dataclass
from typing import Any, Dict, List, Optional

from textual.app import App, ComposeResult
from textual.containers import Horizontal, Vertical
from textual.message import Message
from textual.reactive import reactive
from textual.widgets import (
    Button,
    DataTable,
    Footer,
    Header,
    Input,
    Label,
    Log,
    Select,
```

```
    Static,
)

EE_BIN = "ee"


def safe_site_name(name: str) -> str:
    name = (name or "").strip()
    # Minimal guardrails; EE will validate further.
    if not name or " " in name or "/" in name:
        raise ValueError("Invalid site name.")
    return name


async def run_ee(args: List[str], timeout: Optional[int] = None) -> Dict[str, Any]:
    """
    Run ee command and capture stdout/stderr. Returns dict with rc/stdout/stderr.
    """
    cmd = [EE_BIN] + args
    proc = await asyncio.create_subprocess_exec(
        *cmd,
        stdout=asyncio.subprocess.PIPE,
        stderr=asyncio.subprocess.PIPE,
    )
    try:
        stdout, stderr = await asyncio.wait_for(proc.communicate(), timeout=timeout)
    except asyncio.TimeoutError:
        proc.kill()
        raise
    return {
        "rc": proc.returncode,
        "stdout": (stdout or b"").decode(errors="replace"),
        "stderr": (stderr or b"").decode(errors="replace"),
        "cmd": " ".join(shlex.quote(c) for c in cmd),
    }


async def stream_ee(args: List[str], on_line, on_done) -> None:
    """
    Run ee and stream output lines to callback.
```

```
    """
    cmd = [EE_BIN] + args
    proc = await asyncio.create_subprocess_exec(
        *cmd,
        stdout=asyncio.subprocess.PIPE,
        stderr=asyncio.subprocess.STDOUT,
    )
    await on_line(f"$ {' '.join(shlex.quote(c) for c in cmd)}\n")
    assert proc.stdout is not None
    async for raw in proc.stdout:
        await on_line(raw.decode(errors="replace"))
    rc = await proc.wait()
    await on_done(rc)


@dataclass
class SiteRow:
    name: str
    type: str
    enabled: str
    ssl: str


class StatusBar(Static):
    text = reactive("Ready.")

    def render(self) -> str:
        return f"**Status:** {self.text}"


class EETui(App):
    CSS = """
    Screen { layout: vertical; }
    #main { height: 1fr; }
    #left { width: 1fr; }
    #right { width: 1fr; }
    #actions Button { margin: 1 1; }
    #form Input, #form Select { margin: 1 0; }
    #log { height: 1fr; border: round $surface; }
    #table { height: 1fr; border: round $surface; }
```

```
#info { height: 1fr; border: round $surface; padding: 1; }
"""

BINDINGS = [
    ("q", "quit", "Quit"),
    ("r", "refresh", "Refresh Sites"),
]

# UI state
sites: List[SiteRow] = []
selected_site: Optional[str] = reactive(None)

def compose(self) -> ComposeResult:
    yield Header(show_clock=True)
    with Horizontal(id="main"):
        with Vertical(id="left"):
            yield Label("Websites □□", id="title_left")
            table = DataTable(id="table", zebra_stripes=True)
            table.add_columns("Name", "Type", "Enabled", "SSL")
            yield table
            yield StatusBar(id="status")
        with Vertical(id="right"):
            yield Label("Actions ⚙□", id="title_right")
            with Horizontal(id="actions"):
                yield Button("Refresh", id="btn_refresh", variant="primary")
                yield Button("Info", id="btn_info")
                yield Button("Enable", id="btn_enable")
                yield Button("Disable", id="btn_disable")
                yield Button("Delete", id="btn_delete", variant="error")

            yield Label("Create / Update □")
            with Vertical(id="form"):
                yield Input(placeholder="site name (e.g. example.com)", id="in_name")
                yield Select(
                    options=[
                        ("html", "html"),
                        ("wp", "wp"),
                    ],
                    prompt="site type",
                    id="sel_type",
```

```
                )
                yield Select(
                    options=[
                        ("none", "none"),
                        ("le", "le (Let's Encrypt)"),
                        ("self", "self (self-signed)"),
                    ],
                    prompt="SSL (optional)",
                    id="sel_ssl",
                )
                with Horizontal():
                    yield Button("Create Site", id="btn_create", variant="success")
                    yield Button("Set SSL", id="btn_setssl")

            yield Label("Output □□")
            yield Log(id="log", highlight=True, auto_scroll=True)

            yield Label("Site Info □□")
            yield Static("", id="info")

    yield Footer()

async def on_mount(self) -> None:
    await self.refresh_sites()

def _status(self, msg: str) -> None:
    self.query_one(StatusBar).text = msg

def _log(self, text: str) -> None:
    self.query_one(Log).write(text.rstrip("\n"))

async def refresh_sites(self) -> None:
    self._status("Refreshing sites…")
    self._log("Refreshing site list…")
    res = await run_ee(["site", "list", "--format=json"])
    if res["rc"] != 0:
        self._status("Failed to refresh.")
        self._log(res["stderr"] or res["stdout"])
        return
```

```
        try:
            data = json.loads(res["stdout"] or "[]")
        except json.JSONDecodeError:
            self._status("Failed to parse site list.")
            self._log(res["stdout"])
            return

        sites: List[SiteRow] = []
        for item in data:
            # Fields vary slightly by EE versions; keep it tolerant.
            name = str(item.get("name") or item.get("site") or "")
            typ = str(item.get("type") or "")
            enabled = str(item.get("enabled") or item.get("status") or "")
            ssl = str(item.get("ssl") or item.get("https") or "")
            if name:
                sites.append(SiteRow(name=name, type=typ, enabled=enabled, ssl=ssl))

        self.sites = sorted(sites, key=lambda s: s.name)

        table = self.query_one(DataTable)
        table.clear()
        for s in self.sites:
            table.add_row(s.name, s.type, s.enabled, s.ssl, key=s.name)

        self._status(f"Loaded {len(self.sites)} site(s).")

    async def action_refresh(self) -> None:
        await self.refresh_sites()

    async def on_data_table_row_selected(self, event: DataTable.RowSelected) -> None:
        self.selected_site = str(event.row_key)

    async def _stream_command(self, args: List[str]) -> int:
        self.query_one(Log).write("")  # spacer
        self._status("Running command…")
        rc_holder = {"rc": 1}

        async def on_line(line: str):
            self._log(line)
```

```
    async def on_done(rc: int):
        rc_holder["rc"] = rc
        self._status(f"Done (rc={rc}).")

    await stream_ee(args, on_line, on_done)
    return rc_holder["rc"]

def _get_selected(self) -> str:
    if not self.selected_site:
        raise ValueError("No site selected.")
    return safe_site_name(self.selected_site)

async def on_button_pressed(self, event: Button.Pressed) -> None:
    bid = event.button.id

    try:
        if bid == "btn_refresh":
            await self.refresh_sites()

        elif bid == "btn_info":
            site = self._get_selected()
            rc = await self._stream_command(["site", "info", site])
            if rc == 0:
                res = await run_ee(["site", "info", site])
                self.query_one(Static, id="info").update(
                    f"**{site}**\n\n```\n{(res['stdout'] or '').strip()}\n```"
                )

        elif bid == "btn_enable":
            site = self._get_selected()
            rc = await self._stream_command(["site", "enable", site])
            if rc == 0:
                await self.refresh_sites()

        elif bid == "btn_disable":
            site = self._get_selected()
            rc = await self._stream_command(["site", "disable", site])
            if rc == 0:
                await self.refresh_sites()
```

```
            elif bid == "btn_delete":
                site = self._get_selected()
                # Lightweight “confirmation” via status prompt style:
                self._status(f"Deleting {site}… (no undo)")
                rc = await self._stream_command(["site", "delete", site, "--yes"])
                if rc == 0:
                    self.selected_site = None
                    self.query_one(Static, id="info").update("")
                    await self.refresh_sites()

            elif bid == "btn_create":
                name = safe_site_name(self.query_one(Input, id="in_name").value)
                site_type = self.query_one(Select, id="sel_type").value or "html"
                ssl = self.query_one(Select, id="sel_ssl").value or "none"

                args = ["site", "create", name, f"--type={site_type}"]
                if ssl != "none":
                    args.append(f"--ssl={ssl}")

                # For WP, EE may prompt unless defaults are allowed; you can add fields later.
                rc = await self._stream_command(args)
                if rc == 0:
                    await self.refresh_sites()

            elif bid == "btn_setssl":
                site = self._get_selected()
                ssl = self.query_one(Select, id="sel_ssl").value or "none"
                if ssl == "none":
                    raise ValueError("Select an SSL option (le/self) first.")
                rc = await self._stream_command(["site", "update", site, f"--ssl={ssl}"])
                if rc == 0:
                    await self.refresh_sites()

        except Exception as e:
            self._status(f"Error: {e}")
            self._log(f"Error: {e}")


if __name__ == "__main__":
    EETui().run()
```

Make it executable:

```
chmod +x ee-tui.py
```

Run it:

```
sudo ./ee-tui.py
```

---

3) What you'll likely want to add next □□

1. **WordPress "create" form fields** (title/admin user/email/pass, multisite, cache/redis, php version).
2. **Site clean / cache controls**
   - `ee site clean <site> --page` / `--object`
3. **SSL renew button**
   - `ee site ssl <site> --force`
4. **Logs viewer / service status**
   - Shell into containers: `ee shell <site> --service=nginx|php`
5. **Cron management panel**
   - `ee cron list/create/update/delete`

---

4) A couple of questions so I tailor it to "everything" □

1. Which EasyEngine major version are you on (**v4** or **v5+**)? (`ee --version`)
2. Which site types do you use besides `html`/`wp` (e.g., `php`, `proxy`, etc.)?
3. Do you want the app to support **remote DB**, **WP multisite**, and **Redis cache toggles** from the UI?

If you answer those, I can expand this into a fuller "control panel" TUI with tabs (Sites / WP / SSL / Cron / Shell), better confirmations, and safer privilege handling.

Revision #2
Created 2026-04-17 06:01:51 UTC by art10m
Updated 2026-04-17 06:09:29 UTC by art10m