
Dloper Local-OS – Specification for VS Code / Codex

IMPORTANT FOR CODEX
- Do not change the folder structure.
- Do not rename files.
- Generate each file exactly as specified (paths + content).
- Use Node.js for the backend dashboard.
- Assume this project runs on Raspberry Pi OS Lite on a Raspberry Pi 5.
- This document defines the Free Edition of Dloper Local-OS (static web hosting + web file editor).
- Pro features are described architecturally only (no code needed yet).

---

1. Product overview

1.1 Dloper Local-M1

Dloper Local-M1 is a compact, energy-efficient on-prem web server based on Raspberry Pi 5 hardware with active cooling. It runs Dloper Local-OS, which is a custom OS layer on top of Raspberry Pi OS Lite.

Goals:
- Host websites and simple tools inside the local network.
- Provide a web dashboard that:
- Lists configured sites.
- Lets users browse site files and edit them in a code-editor style interface.
- Lets users preview their site in a browser.

1.2 Editions

We plan three editions:
- Dloper OS Lite (Free)
- Static website hosting from /srv/sites/<site-id>/www
- Local dashboard (no login for now)
- File browser + editor
- No HTTPS, no analytics, no tunnels, no Docker
- Dloper OS Lite+ (Paid)
- Everything in Lite
- Basic password protection for dashboard and sites
- Basic HTTPS support
- Simple backup/restore
- Dloper OS Professional (Paid)
- Multiple sites with isolation
- Password-protected websites
- HTTPS with automatic certificate management
- Analytics dashboard
- ProSupport bundle export
- Automatic OS updates
- Optional Cloudflare Tunnel / remote access
- License activation (product key + email)

For this Codex build:
Only implement the Lite / Free Edition features. Pro and Lite+ are future work.

---

2. Target platform & requirements

- Hardware: Raspberry Pi 5
- OS base: Raspberry Pi OS Lite (64-bit)
- Package manager: apt
- System services: systemd
- Web server: Node.js app on port 8080, optionally behind Nginx on port 80.

2.1 System packages to install

- nginx
- git
- curl
- nodejs
- npm

2.2 Node dependencies (dashboard backend)

Install via npm install:

- express
- cors
- body-parser
- js-yaml

---

3. Folder structure

The project root is dloper-local-os/. Codex must respect this structure exactly:

```
dloper-local-os/
  README.md

  dashboard/
    backend/
      package.json
      server.js
    frontend/
      index.html
      style.css
      app.js

  services/
    nginx/
      dloper-dashboard.conf
    systemd/
      dloper-dashboard.service

  setup/
    bootstrap.sh
    install_nginx.sh
    install_docker.sh
    install_tunnel.sh

  site-config/
    example-site.yml
```

---

4. Installation & boot flow (Free Edition)

1. User flashes a standard Raspberry Pi OS Lite image to an SD card.

2. After first boot and network setup, user SSHs into the Pi.
3. User clones this repo to /opt/dloper-local-os or copies it there.
4. User runs:

```
cd /opt/dloper-local-os
sudo bash setup/bootstrap.sh
```

5. bootstrap.sh:
   - Updates system packages.
   - Installs Nginx, Node.js and required packages.
   - Installs the dloper-dashboard.service systemd unit.
   - Installs Nginx reverse proxy config.
   - Enables the dashboard service (but does not start it automatically in the script).
6. User creates a demo site:

```
sudo mkdir -p /srv/sites/example/www
echo "<h1>Hello from Dloper Local OS</h1>" | sudo tee /srv/sites/example/www/index.html
```

7. User runs:

```
sudo systemctl daemon-reload
sudo systemctl enable dloper-dashboard.service
sudo systemctl start dloper-dashboard.service
```

8. User opens a browser at http://<pi-ip>/ or http://<pi-ip>:8080 to access the dashboard.

---

5. Tasks for Codex

Codex must:

1. Create all files listed in this document with exact paths and exact content.
2. Ensure all scripts have the correct shebang (#!/usr/bin/env bash).
3. Ensure server.js uses the API paths and behaviour described.
4. Ensure Node backend serves the static frontend from dashboard/frontend.
5. Ensure file editor behaviour:
   - Click site → loads site file tree.
   - Click a directory → open that directory.
   - Click a file → load contents into text editor.
   - Save button → PUT to API to write new file contents.
   - Preview button → opens /sites/<site-id>/… view in a new tab.
6. Not implement any login/auth, analytics, tunnelling or license system yet.

---

6. File manifest and content

Below is every file Codex must create.

6.1 README.md

Path: README.md

~~~~
# Dloper Local OS (Free Edition)

Minimal local OS layer for Raspberry Pi:
- Static website hosting from `/srv/sites/<site-id>/www`
- Web dashboard (port 8080) to:
  - List configured sites
  - Browse folders/files
  - Edit file contents and save
  - Preview the website

## Structure

- dashboard/
  - backend/  -> Node.js API
  - frontend/ -> HTML/CSS/JS dashboard UI
- services/
  - nginx/    -> Reverse proxy config (optional)
  - systemd/  -> Service definitions
- setup/
  - bootstrap.sh       -> Main installer
  - install_nginx.sh   -> Nginx setup
  - install_docker.sh  -> (currently stub)
  - install_tunnel.sh  -> (currently stub)
- site-config/
  - example-site.yml   -> Example site definition

## Basic usage (on the Pi)

```bash
# 1. Clone or copy this repo to /opt
sudo mkdir -p /opt
cd /opt
sudo git clone <your-repo-url> dloper-local-os
cd dloper-local-os

# 2. Run bootstrap
sudo bash setup/bootstrap.sh

# 3. Create example site root
sudo mkdir -p /srv/sites/example/www
echo "<h1>Hello from Dloper Local OS</h1>" | sudo tee /srv/sites/example/www/index.html

# 4. Enable and start the dashboard
sudo systemctl daemon-reload
sudo systemctl enable dloper-dashboard.service
sudo systemctl start dloper-dashboard.service

Then open: http://<pi-ip>:8080 or http://<pi-ip>/ in your browser.

---

### 6.2 Setup scripts

#### 6.2.1 `setup/bootstrap.sh`

**Path:** `setup/bootstrap.sh`

```bash
#!/usr/bin/env bash
set -e
~~~~

```
# Simple bootstrap for Dloper Local OS (Free Edition)

if [[ $EUID -ne 0 ]]; then
  echo "Please run as root: sudo bash setup/bootstrap.sh"
  exit 1
fi

SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
REPO_ROOT="$(cd "${SCRIPT_DIR}/.." && pwd)"

echo "[Dloper] Updating apt..."
apt update -y
apt upgrade -y

echo "[Dloper] Installing base packages (git, curl, nginx, nodejs, npm)..."
apt install -y git curl nginx nodejs npm

echo "[Dloper] Creating directories..."
mkdir -p /srv/sites
mkdir -p /var/log/dloper

# Install Node dependencies for dashboard backend
echo "[Dloper] Installing Node dependencies..."
cd "${REPO_ROOT}/dashboard/backend"
npm install --production

# Copy systemd service
echo "[Dloper] Installing systemd service..."
cp "${REPO_ROOT}/services/systemd/dloper-dashboard.service" /etc/systemd/system/

# Nginx config (reverse proxy to dashboard)
echo "[Dloper] Installing Nginx config..."
cp "${REPO_ROOT}/services/nginx/dloper-dashboard.conf" /etc/nginx/sites-available/dloper-
dashboard.conf
ln -sf /etc/nginx/sites-available/dloper-dashboard.conf /etc/nginx/sites-enabled/dloper-
dashboard.conf

# Disable default nginx site if present
if [ -f /etc/nginx/sites-enabled/default ]; then
  rm /etc/nginx/sites-enabled/default
fi

nginx -t && systemctl reload nginx

echo "[Dloper] Enabling dashboard service..."
systemctl daemon-reload
systemctl enable dloper-dashboard.service

echo "[Dloper] Bootstrap complete!
- Dashboard service: dloper-dashboard.service
- Preview: http://<pi-ip>:8080
- Nginx proxy: http://<pi-ip>/ (port 80)
"
```

———

6.2.2 setup/install_nginx.sh
Path: setup/install_nginx.sh

```
#!/usr/bin/env bash
set -e

if [[ $EUID -ne 0 ]]; then
  echo "Please run as root: sudo bash setup/install_nginx.sh"
  exit 1
fi

SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
REPO_ROOT="$(cd "${SCRIPT_DIR}/.." && pwd)"

apt update -y
apt install -y nginx

cp "${REPO_ROOT}/services/nginx/dloper-dashboard.conf" /etc/nginx/sites-available/dloper-
dashboard.conf
ln -sf /etc/nginx/sites-available/dloper-dashboard.conf /etc/nginx/sites-enabled/dloper-
dashboard.conf

# disable default
if [ -f /etc/nginx/sites-enabled/default ]; then
  rm /etc/nginx/sites-enabled/default
fi

nginx -t && systemctl reload nginx

echo "[Dloper] Nginx installed and configured."
```

6.2.3 setup/install_docker.sh (stub)
Path: setup/install_docker.sh

```
#!/usr/bin/env bash
set -e

echo "[Dloper] Docker is not required for the free edition (static hosting only)."
echo "[Dloper] You can install Docker later if you want to extend functionality."
```

6.2.4 setup/install_tunnel.sh (stub)
Path: setup/install_tunnel.sh

```
#!/usr/bin/env bash
set -e

echo "[Dloper] Cloudflare Tunnel / remote access is disabled in the free edition."
echo "[Dloper] The dashboard and sites are only available on the local network."
```

6.3 Systemd service

6.3.1 services/systemd/dloper-dashboard.service
Path: services/systemd/dloper-dashboard.service

```
[Unit]
Description=Dloper Local OS Dashboard
After=network.target

[Service]
Type=simple
User=pi
WorkingDirectory=/opt/dloper-local-os/dashboard/backend
ExecStart=/usr/bin/node server.js
Restart=always
RestartSec=5
Environment=NODE_ENV=production

[Install]
WantedBy=multi-user.target
```

If the repo is placed elsewhere, the WorkingDirectory may need updating, but default assumption is /opt/dloper-local-os.

6.4 Nginx config

6.4.1 services/nginx/dloper-dashboard.conf
Path: services/nginx/dloper-dashboard.conf

```
server {
   listen 80 default_server;
   listen [::]:80 default_server;

   server_name _;

   # Proxy all requests to the Node.js dashboard (port 8080)
   location / {
      proxy_pass http://127.0.0.1:8080;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Forwarded-Proto $scheme;
   }
}
```

6.5 Site config example

6.5.1 site-config/example-site.yml
Path: site-config/example-site.yml

```
id: example
name: Example Website
description: |
  A demo site hosted by Dloper Local OS (free edition).
root: /srv/sites/example/www
```

6.6 Dashboard backend (Node.js API)

6.6.1 dashboard/backend/package.json
Path: dashboard/backend/package.json

```
{
  "name": "dloper-local-dashboard",
  "version": "0.1.0",
  "description": "Dloper Local OS free edition dashboard backend",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "express": "^4.19.0",
    "js-yaml": "^4.1.0"
  }
}
```

———

6.6.2 dashboard/backend/server.js
Path: dashboard/backend/server.js

```
const express = require("express");
const cors = require("cors");
const bodyParser = require("body-parser");
const fs = require("fs");
const path = require("path");
const yaml = require("js-yaml");

const app = express();
const PORT = 8080;

// Paths
const BACKEND_DIR = __dirname;
const REPO_ROOT = path.join(BACKEND_DIR, "..", "..");
const SITE_CONFIG_DIR = path.join(REPO_ROOT, "site-config");

// Middleware
app.use(cors());
app.use(bodyParser.text({ type: "*/*", limit: "5mb" })); // for editing text files
app.use(express.static(path.join(REPO_ROOT, "dashboard", "frontend"))); // serve UI

// ---------- Helpers ----------

function loadSites() {
  const files = fs
    .readdirSync(SITE_CONFIG_DIR)
    .filter((f) => f.endsWith(".yml") || f.endsWith(".yaml"));
  return files.map((file) => {
    const configPath = path.join(SITE_CONFIG_DIR, file);
    const data = yaml.load(fs.readFileSync(configPath, "utf8"));
    return {
      id: data.id,
      name: data.name || data.id,
      description: data.description || "",
      root: data.root,
    };
  });
}

function getSiteById(siteId) {
  const sites = loadSites();
  return sites.find((s) => s.id === siteId);
}

function safeResolve(siteRoot, requestedPath) {
  const rel = requestedPath || "/";
  const fullPath = path.normalize(path.join(siteRoot, rel));
  if (!fullPath.startsWith(siteRoot)) {
    throw new Error("Invalid path");
  }
  return fullPath;
}

function listDirectory(siteRoot, relPath = "/") {
  const dirPath = safeResolve(siteRoot, relPath);
  const entries = fs.readdirSync(dirPath, { withFileTypes: true });

  return entries.map((entry) => ({
    name: entry.name,
    type: entry.isDirectory() ? "dir" : "file",
    path: path.posix.join(relPath === "/" ? "" : relPath, entry.name),
  }));
}

// ---------- API Routes ----------

// List all sites
app.get("/api/sites", (req, res) => {
  try {
    const sites = loadSites();
    res.json(sites);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "Failed to load sites" });
  }
});

// List directory tree for a site (one level at a time)
app.get("/api/sites/:siteId/tree", (req, res) => {
  try {
    const site = getSiteById(req.params.siteId);
    if (!site) return res.status(404).json({ error: "Site not found" });

    const relPath = req.query.path || "/";
    const data = listDirectory(site.root, relPath);
    res.json({ items: data, path: relPath });
  } catch (err) {
    console.error(err);
    res.status(400).json({ error: err.message || "Invalid path" });
  }
});

// Read file contents
app.get("/api/sites/:siteId/file", (req, res) => {
  try {
    const site = getSiteById(req.params.siteId);
    if (!site) return res.status(404).json({ error: "Site not found" });

    const relPath = req.query.path;
    if (!relPath) return res.status(400).json({ error: "Missing path" });

    const fullPath = safeResolve(site.root, relPath);
    const content = fs.readFileSync(fullPath, "utf8");
    res.type("text/plain").send(content);
  } catch (err) {
    console.error(err);
    res.status(400).json({ error: err.message || "Failed to read file" });
  }
});

// Save file contents
app.put("/api/sites/:siteId/file", (req, res) => {
  try {
    const site = getSiteById(req.params.siteId);
    if (!site) return res.status(404).json({ error: "Site not found" });

    const relPath = req.query.path;
    if (!relPath) return res.status(400).json({ error: "Missing path" });

    const fullPath = safeResolve(site.root, relPath);
    fs.writeFileSync(fullPath, req.body ?? "", "utf8");
    res.json({ ok: true });
  } catch (err) {
    console.error(err);
    res.status(400).json({ error: err.message || "Failed to write file" });
  }
});

// Preview site files (static hosting)
app.use("/sites/:siteId", (req, res, next) => {
  const siteId = req.params.siteId;
  const site = getSiteById(siteId);
  if (!site) return res.status(404).send("Site not found");

  const siteRoot = site.root;
  const staticMiddleware = express.static(siteRoot);
  return staticMiddleware(req, res, next);
});

// Fallback to index.html of frontend
app.get("*", (req, res) => {
  res.sendFile(path.join(REPO_ROOT, "dashboard", "frontend", "index.html"));
});

app.listen(PORT, () => {
  console.log(`Dloper dashboard listening on port ${PORT}`);
});
```

6.7 Dashboard frontend

6.7.1 dashboard/frontend/index.html

Path: dashboard/frontend/index.html

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>Dloper Local OS – Dashboard</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <div id="app">
      <header class="topbar">
        <h1>Dloper Local OS</h1>
        <span class="subtitle">Free Edition · Static Hosting</span>
      </header>

      <div class="main">
        <aside class="sidebar">
          <h2>Sites</h2>
          <ul id="sites-list"></ul>

          <h2>Files</h2>
          <div id="current-path"></div>
          <ul id="files-list"></ul>
        </aside>

        <section class="editor">
          <div class="editor-header">
            <span id="editor-filename">(no file selected)</span>
            <div class="editor-actions">
              <button id="btn-preview" disabled>Preview</button>
              <button id="btn-save" disabled>Save</button>
            </div>
          </div>
          <textarea
            id="editor-text"
            spellcheck="false"
            disabled
          ></textarea>
          <div id="status-bar"></div>
        </section>
      </div>
    </div>

    <script src="app.js"></script>
  </body>
</html>
```

———

6.7.2 dashboard/frontend/style.css
Path: dashboard/frontend/style.css

```css
* {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
  font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI",
    sans-serif;
}

body {
  background: #0f172a;
  color: #e5e7eb;
  height: 100vh;
}

#app {
  display: flex;
  flex-direction: column;
  height: 100vh;
}

.topbar {
  display: flex;
  align-items: baseline;
  gap: 12px;
  padding: 12px 20px;
  background: #020617;
  border-bottom: 1px solid #1e293b;
}

.topbar h1 {
  font-size: 18px;
  font-weight: 600;
}

.topbar .subtitle {
  font-size: 13px;
  color: #9ca3af;
}

.main {
  display: flex;
  flex: 1;
  min-height: 0;
}

/* Sidebar */

.sidebar {
  width: 260px;
  background: #020617;
  border-right: 1px solid #1e293b;
  padding: 10px;
  display: flex;
  flex-direction: column;
  gap: 10px;
  overflow: auto;
}

.sidebar h2 {
  font-size: 13px;
  text-transform: uppercase;
  letter-spacing: 0.08em;
  color: #6b7280;
  margin-bottom: 4px;
}
```

```
#sites-list,
#files-list {
  list-style: none;
}

#sites-list li,
#files-list li {
  padding: 6px 8px;
  margin-bottom: 2px;
  border-radius: 4px;
  font-size: 13px;
  cursor: pointer;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

#sites-list li:hover,
#files-list li:hover {
  background: #111827;
}

#sites-list li.active,
#files-list li.active {
  background: #1d283a;
}

/* Path */

#current-path {
  font-size: 11px;
  color: #9ca3af;
  margin-bottom: 4px;
  word-break: break-all;
}

/* Editor */

.editor {
  flex: 1;
  display: flex;
  flex-direction: column;
  min-width: 0;
}

.editor-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 8px 14px;
  border-bottom: 1px solid #1e293b;
  background: #020617;
}

#editor-filename {
  font-size: 13px;
  color: #e5e7eb;
}

.editor-actions button {
  font-size: 13px;
  padding: 5px 12px;
  margin-left: 6px;
  border-radius: 4px;
  border: none;
  cursor: pointer;
  background: #334155;
  color: #e5e7eb;
}

.editor-actions button:disabled {
  opacity: 0.4;
  cursor: default;
}

.editor-actions button:hover:not(:disabled) {
  background: #2563eb;
}

#editor-text {
  flex: 1;
  width: 100%;
  border: none;
  outline: none;
  resize: none;
  padding: 10px 14px;
  font-family: "JetBrains Mono", "SF Mono", Menlo, Monaco, Consolas,
    monospace;
  font-size: 13px;
  background: #020617;
  color: #e5e7eb;
}

#status-bar {
  height: 22px;
  font-size: 11px;
  display: flex;
  align-items: center;
  padding: 0 10px;
  border-top: 1px solid #1e293b;
  background: #020617;
  color: #9ca3af;
}

/* Small screens */

@media (max-width: 800px) {
  .main {
    flex-direction: column;
  }
  .sidebar {
    width: 100%;
    min-height: 200px;
  }
}
```

6.7.3 dashboard/frontend/app.js

Path: dashboard/frontend/app.js

```
const sitesListEl = document.getElementById("sites-list");
const filesListEl = document.getElementById("files-list");
const currentPathEl = document.getElementById("current-path");
const editorTextEl = document.getElementById("editor-text");
const editorFilenameEl = document.getElementById("editor-filename");
const statusBarEl = document.getElementById("status-bar");
const btnSave = document.getElementById("btn-save");
const btnPreview = document.getElementById("btn-preview");

let sites = [];
let currentSite = null;
let currentPath = "/";
let currentFilePath = null;

async function api(path, options = {}) {
  const res = await fetch(path, options);
  if (!res.ok) {
    const text = await res.text();
    throw new Error(text || `Request failed: ${res.status}`);
  }
  const contentType = res.headers.get("content-type") || "";
  if (contentType.includes("application/json")) {
    return res.json();
  }
  return res.text();
}

function setStatus(msg) {
  statusBarEl.textContent = msg || "";
}

function renderSites() {
  sitesListEl.innerHTML = "";
  sites.forEach((site) => {
    const li = document.createElement("li");
    li.textContent = site.name;
    li.dataset.id = site.id;
    if (currentSite && currentSite.id === site.id) li.classList.add("active");
    li.addEventListener("click", () => {
      currentSite = site;
      currentPath = "/";
      currentFilePath = null;
      renderSites();
      loadDirectory("/");
      clearEditor();
    });
    sitesListEl.appendChild(li);
  });
}

function renderFiles(items) {
  filesListEl.innerHTML = "";

  // ".." up one level
  if (currentPath !== "/") {
    const liUp = document.createElement("li");
    liUp.textContent = "..";
    liUp.addEventListener("click", () => {
      const parts = currentPath.split("/").filter(Boolean);
      parts.pop();
      const parent = "/" + parts.join("/");
      loadDirectory(parent === "//" ? "/" : parent || "/");
    });
    filesListEl.appendChild(liUp);
  }

  items.forEach((item) => {
    const li = document.createElement("li");
    li.textContent = item.name;
    li.dataset.path = item.path;

    const badge = document.createElement("span");
    badge.textContent = item.type === "dir" ? "DIR" : "FILE";
    badge.style.fontSize = "10px";
    badge.style.opacity = 0.6;
    li.appendChild(badge);

    li.addEventListener("click", () => {
      if (item.type === "dir") {
        loadDirectory(item.path);
      } else {
        openFile(item.path);
      }
    });

    if (item.path === currentFilePath) {
      li.classList.add("active");
    }

    filesListEl.appendChild(li);
  });
}

function clearEditor() {
  editorTextEl.value = "";
  editorTextEl.disabled = true;
  btnSave.disabled = true;
  btnPreview.disabled = true;
  editorFilenameEl.textContent = "(no file selected)";
  setStatus("");
}

async function loadSites() {
  try {
    setStatus("Loading sites...");
    sites = await api("/api/sites");
    setStatus("");
    if (sites.length === 0) {
      setStatus("No sites configured yet. Add YAML files in /site-config.");
      return;
    }
    currentSite = sites[0];
    renderSites();
    loadDirectory("/");
  } catch (err) {
    console.error(err);
    setStatus("Failed to load sites.");
  }
```

```
}

async function loadDirectory(path) {
  if (!currentSite) return;
  try {
    setStatus(`Loading ${path}...`);
    const data = await api(
      `/api/sites/${currentSite.id}/tree?path=${encodeURIComponent(path)}`
    );
    currentPath = data.path;
    currentPathEl.textContent = currentPath;
    renderSites();
    renderFiles(data.items);
    setStatus("");
  } catch (err) {
    console.error(err);
    setStatus("Failed to load directory.");
  }
}

async function openFile(filePath) {
  if (!currentSite) return;
  try {
    setStatus(`Opening ${filePath}...`);
    const content = await api(
      `/api/sites/${currentSite.id}/file?path=${encodeURIComponent(filePath)}`
    );
    currentFilePath = filePath;
    editorTextEl.disabled = false;
    btnSave.disabled = false;
    btnPreview.disabled = false;
    editorFilenameEl.textContent = `${currentSite.id}:${filePath}`;
    editorTextEl.value = content;
    setStatus("File loaded.");
  } catch (err) {
    console.error(err);
    setStatus("Failed to open file.");
  }
}

async function saveFile() {
  if (!currentSite || !currentFilePath) return;
  try {
    setStatus("Saving...");
    await api(
      `/api/sites/${currentSite.id}/file?path=${encodeURIComponent(
        currentFilePath
      )}`,
      {
        method: "PUT",
        headers: {
          "Content-Type": "text/plain",
        },
        body: editorTextEl.value,
      }
    );
    setStatus("Saved.");
  } catch (err) {
    console.error(err);
    setStatus("Failed to save file.");
  }
}

function previewFile() {
  if (!currentSite) return;
  const base = `/sites/${encodeURIComponent(currentSite.id)}`;
  const url =
    currentFilePath && currentFilePath.endsWith(".html")
      ? `${base}/${currentFilePath}`
      : base + "/";
  window.open(url, "_blank");
}

btnSave.addEventListener("click", saveFile);
btnPreview.addEventListener("click", previewFile);

// Init
loadSites();
```