Nitro — Web Editor

Nitro is a lightweight web-based code editor that opens a local project folder and lets users browse files, read/edit/save files, and perform basic file operations (create folder/file, delete). This document describes how Nitro works, the frontend and backend APIs, UI components, install/run instructions, security notes, and a short roadmap.

---

1. Overview

Nitro is designed to expose a local project folder through a small Flask backend and a React frontend. The flow:

1. User copies a project path from their system (e.g. `C:\Users\me\projects\my-app`).
2. User pastes the path into the editor input. Nitro posts the path to the backend and sets it as the working directory.
3. The Explorer shows files and folders. Clicking folders expands to show children (lazy loaded). Clicking files reads file content into the editor.
4. The editor (Monaco) displays content with syntax highlighting. Users can edit and save — which writes content back to disk.
5. Users can create files/folders and delete items using explorer actions.

---

2. High-level architecture

- **Frontend (React)**
- Explorer UI to navigate the filesystem.
- Recursive file list component that supports expand/collapse and per-item actions.
- File editor using Monaco Editor for syntax highlighting and editing.
- A small context (`FileTabContext`) holds the currently opened file and content.

- **Backend (Flask)**

- Class-based API `FileManagerAPI` with routes to select working dir, list directory, read/write files, create/delete files and folders.
- CORS enabled for local development.

---

3. Backend — API endpoints

All endpoints expect JSON and return JSON. Example base: `http://127.0.0.1:5000`

- `POST /choose` — set working directory
- Request: `{ "path": "C:/path/to/project" }`
- Response: `{ "msg": "Working directory is updated", "workingdir": "..." }`
- `GET /cd` — list files/folders in current working directory
- Response: `{ "msg": "Directory exists", "dir": "...", "data": [ {type, name, path, extension}, ... ] }`
- `POST /selectFolder` — list children of a given folder (used for lazy expansion)
- Request: `{ "path": "/full/folder/path" }`
- Response: `{ "msg": "Directory exists", "data": [...] }`
- `POST /ReadFile` — read file contents
- Request: `{ "path": "/full/file/path" }`
- Response: `{ "msg":"File read successfully","path":"...","content":"..." }`
- `POST /WriteFile` — overwrite file with content
- Request: `{ "path": "/full/file/path", "content": "new file text" }`
- Response: `{ "msg":"File written successfully","path":"...","size":123 }`
- `POST /makefile` — create an empty file
- Request: `{ "rootdir": "/path/to/dir","filename":"newfile.txt" }`
- `POST /makefolder` — create folder
- Request: `{ "rootdir": "/path/to/dir","foldername":"newfolder" }`
- `POST /delete` — delete file or directory (recursive)
- Request: `{ "path": "/full/path", "name": "name" }`

  Tip: Keep these requests JSON and use `Content-Type: application/json` so Flask `request.get_json()` works without 415 errors.

---

4. Frontend components

- **Explore.jsx** — top-level explorer layout and API calls (`/choose`, `/cd`, `fetchDirectory`).
- **ListDist.jsx** — recursive list that renders file rows, handles expand/collapse, fetches children via `/selectFolder`, and shows file action buttons (add, folder-add, rename, delete). It calls `openFile` when a file is selected.
- **File.jsx** — editor pane using `@monaco-editor/react`. Handles `value`, `onChange`, theme switching, and calls `WriteFile` to save.
- **ToolBar.jsx** — optional toolbar with theme selector, save shortcut hints, etc.
- **ExtensionIcon.jsx** — small helper to return icons per extension.
- **FileTabContext** — React context storing `fileopen`, `fileData`, and setter functions.

---

5. UI behavior / UX details

- **Explorer**
- Input at top for pasted project path.
- Buttons to create file/folder (in the selected working dir).
- Clicking a folder toggles expand/collapse — children fetched lazily by `/selectFolder`.

- Hovering a file shows action icons (edit, delete, add).

- **Editor**

- Monaco Editor with theme support (you can register multiple themes via `monaco.editor.defineTheme(...)` and `monaco.editor.setTheme('name')`).
- Save button and `Ctrl+S` keyboard shortcut to call `/WriteFile`.
- Tab header shows opened file name + icon; clicking close clears `fileopen`.

---

6. Minimal example snippets

Frontend: read a file (axios)

```
// file = { path: '/abs/path/file.js' }
axios.post('http://127.0.0.1:5000/ReadFile', { path: file.path })
  .then(res => setFileData(res.data.content))
  .catch(err => console.error(err));
```

Frontend: write/save a file (fetch)

```
await fetch('http://127.0.0.1:5000/WriteFile', {
  method: 'POST',
  headers: {'Content-Type':'application/json'},
  body: JSON.stringify({ path: fileopen.path, content: fileData })
});
```

Backend: safe write (overwrite) pseudocode

```
def WriteFile(self):
    try:
        filedata = request.get_json()
        target_path = filedata.get('path')
        content = filedata.get('content', '')
        if os.path.isfile(target_path):
            with open(target_path, 'w', encoding='utf-8') as f:
                f.write(content)
            return jsonify({'msg':'File written
successfully','path':target_path}),200
        return jsonify({'msg':'File does not exist','path':target_path}),404
    except Exception as e:
        return jsonify({'msg':'Error writing file','error':str(e)}),500
```

---

7. Security & privacy notes (important)

- **This tool accesses the user filesystem.** Running Nitro as-is exposes local files to the frontend and any site that can reach your server. Only run Nitro locally on your machine.
- **Do not deploy to public servers** unless you add strict authentication and sandboxing. Exposing `WriteFile` + `Delete` endpoints on a public server is dangerous.
- Consider limiting the allowed root paths or asking the user to confirm explicit directories.

---

8. Running locally (quick start)

1. Backend (Flask)
2. Create a virtualenv, `pip install flask flask-cors`
3. Save the `FileManagerAPI` class as `app.py` and run `python app.py` (or `FLASK_APP=app.py flask run`).
4. Frontend (React)

5. `npx create-react-app nitro` (or use your existing project)
6. Install: `npm i axios @monaco-editor/react`
7. Add components and context, then `npm start`.
8. Open `http://localhost:3000`, paste a project path, and start exploring.

---

9. Feature roadmap (suggested)

- Multiple tabs with persistent tab state.
- File rename endpoint + UI.
- Search-in-files with recursive grep.
- File watchers to auto-refresh explorer when project files change.
- Permission checks & read-only mode.
- Optional ZIP export / import.

---

10. Design language / Theme

- Editor name: **Nitro** (branding: energetic, neon accent).
- Default editor theme: dark (Monaco `vs-dark`) with optional custom themes (`dracula`, `solarized`, `ocean`).

---

11. Quick checklist before sharing Nitro with others

- [ ] Remove CORS or restrict origins.
- [ ] Add auth (password or token) if remote access is allowed.
- [ ] Add logging and error handling for disk operations.
- [ ] Add safe path validation to prevent directory traversal attacks.

If you want, I can: - Generate a **README.md** file ready for your repo.
- Produce a **sample UI mockup** (wireframe) for Nitro.
- Export the backend `app.py` and example React components as files in a canvas.

Which of these do you want next?