VIBE-CODING LOCAL TOOL (React + TypeScript Only)
“Multi-Model Collaboration System using OpenRouter (Free Models)”
Complete build document (nothing missed from the chat)

1) Goal
Build a local web app (vibe-coding tool) to generate premium landing pages and full websites using React + TypeScript ONLY (.tsx output).
Core innovation: Use multiple OpenRouter models simultaneously, each thinking differently, then merge outputs to maximize quality.

2) Why OpenRouter
- One API key
- Multiple models access and switching
- Free tier experimentation
- Scalable to paid models later

3) Models to Integrate (from chat)
- qwen/qwen3-coder:free
- google/gemma-3n-e2b-it:free
- tngtech/deepseek-r1t2-chimera:free

4) Best Roles for Each Model
A) qwen/qwen3-coder:free (Code Agent)
Best for React + TSX code generation:
- components, hooks, state, props
- TypeScript interfaces/types
- refactor and fixes

B) google/gemma-3n-e2b-it:free (Copywriter Agent)
Best for conversion copy:
- hero headline/subheadline
- CTA microcopy
- features/pricing/FAQ text

C) tngtech/deepseek-r1t2-chimera:free (Planner/Architect Agent)
Best for reasoning and structure:
- landing page layout plan
- section ordering
- UX improvements
- component breakdown

5) Key Feature: Simultaneous Multi-Model Thinking
Run all agents in parallel:
- Planner output + Copy output + Code output
Then aggregate/merge:
- final premium output
- plus extra improvements beyond prompt

6) Premium Pipeline (Parallel + Merge)
1) User prompt
2) Backend sends prompt to all models in parallel
3) Collect 3 outputs:
  - plan
  - copy

- code
4) Aggregator merges all into final codebase
5) Return final output + intermediate outputs

7) Output Standard
Always output:
- React + TypeScript only (.tsx)
- production-ready, premium UI
- clean component structure

Output formats:
Option 1: Single file (Home.tsx)
Option 2: Multi-file structure (recommended):
- src/pages/Home.tsx
- src/components/Hero.tsx
- src/components/Features.tsx
- src/components/Pricing.tsx
- src/components/FAQ.tsx
- src/components/Footer.tsx

8) Recommended Tech Stack
Frontend:
- React + TypeScript (Vite)
- Tailwind CSS (optional but ideal for landing pages)
- Monaco Editor (in-browser editor)
- Sandpack or iframe preview

Backend:
- FastAPI recommended (clean + async)
- Node.js possible alternative

AI:
- OpenRouter API
- parallel agent calls + aggregator

9) Full Folder Structure
vibe-coding-tool/
■■■ backend/
■ ■■■ app/
■ ■ ■■■ main.py
■ ■ ■■■ core/
■ ■ ■ ■■■ config.py
■ ■ ■ ■■■ cors.py
■ ■ ■■■ api/
■ ■ ■ ■■■ routes_generate.py
■ ■ ■■■ services/
■ ■ ■ ■■■ openrouter_client.py
■ ■ ■ ■■■ agents.py
■ ■ ■ ■■■ orchestrator.py
■ ■ ■ ■■■ postprocessor.py
■ ■ ■■■ schemas/
■ ■ ■ ■■■ generate.py
■ ■ ■■■ utils/

■ ■ ■■■ prompt_templates.py
■ ■ ■■■ validators.py
■ ■■■ requirements.txt
■ ■■■ .env
■
■■■ frontend/
■ ■■■ src/
■ ■ ■■■ components/
■ ■ ■ ■■■ PromptBox.tsx
■ ■ ■ ■■■ ModelChips.tsx
■ ■ ■ ■■■ OutputTabs.tsx
■ ■ ■ ■■■ Editor.tsx
■ ■ ■ ■■■ Preview.tsx
■ ■ ■ ■■■ ExportButton.tsx
■ ■ ■■■ pages/
■ ■ ■ ■■■ Playground.tsx
■ ■ ■■■ services/
■ ■ ■ ■■■ api.ts
■ ■ ■■■ types/
■ ■ ■ ■■■ ai.ts
■ ■ ■■■ App.tsx
■ ■ ■■■ main.tsx
■ ■■■ vite.config.ts
■ ■■■ package.json
■ ■■■ tailwind.config.ts
■
■■■ docs/
■ ■■■ architecture.md
■ ■■■ prompts.md
■ ■■■ api.md
■
■■■ README.md

10) Backend Responsibilities
- store OpenRouter API key securely
- run multi-agent parallel generation
- merge outputs into final premium result
- return plan/copy/code/final outputs to frontend

11) Main API Endpoint
POST /generate

Request example:
```
{
  "prompt": "Build a modern SaaS landing page",
  "tech": "react-ts",
  "agents": ["planner", "copywriter", "coder"],
  "style": "premium",
  "sections": ["hero", "features", "pricing", "faq", "footer"]
}
```

Response example:

```
{
  "plan": "...",
  "copy": "...",
  "code": "...tsx...",
  "final": {
    "files": {
      "src/pages/Home.tsx": "...",
      "src/components/Hero.tsx": "...",
      "src/components/Pricing.tsx": "..."
    }
  }
}
```

12) Parallel Execution Strategy
- call DeepSeek planner
- call Gemma copywriter
- call Qwen coder
- wait for all results
- call aggregator model to merge into final output

13) Prompt Templates (Agent Instructions)
Planner (DeepSeek): UX architect output with structure + component plan + improvements
Copywriter (Gemma): premium conversion text for each section
Coder (Qwen): React TSX-only production UI code

14) Aggregator (Merger) Agent
Input: plan + copy + code outputs
Output: final polished codebase with best layout, best copy, best design, extra improvements.

15) Post-Processing Rules
- remove markdown fences
- ensure valid TSX
- enforce React+TS only
- auto-fix imports
- validate JSX/output structure

16) Frontend UI Layout (Recommended)
Left panel:
- prompt input
- generate/regenerate
- “Make it premium”
- model status indicators

Right panel:
Tabs:
- Final Output
- Plan
- Copy
- Code

Preview:
- live UI preview using sandpack/iframe

17) Model Selector (Optional UX)

- Auto Mode (recommended): uses all agents
- Manual Mode: planner only, copy only, coder only, all together

18) Local Setup Commands
Backend:
cd backend
python -m venv venv
venv\Scripts\activate
pip install -r requirements.txt
uvicorn app.main:app --reload

Frontend:
cd frontend
npm install
npm run dev

19) Security Rules
- Store OpenRouter API key only in backend .env
Example:
OPENROUTER_API_KEY=your_key_here
- Never expose key in frontend

20) Premium Features (Innovation)
- Prompt Enhancer (auto rewrite user prompt into clear specs)
- Style presets (Minimal, Glass, Neon, Modern SaaS, Corporate, Portfolio)
- Improve mode (“make it more premium”, “add animations”, “more modern”)
- Export ZIP of generated project
- Regenerate per-section
- Improve existing code loop

21) Final Summary
You are building a local vibe-coding tool:
- React + TypeScript only
- powered by OpenRouter
- three free models
- parallel + merge architecture
- premium results and extra improvements